Transformers for tabular data

Maria Tikhonova

Outline

- Motivation
- TabTransformer Deep Dive
- Linear Numerical Embeddings
- Piecewise Linear & Periodic Encoding
- AdHoc: TabNet

Outline

- Motivation
- TabTransformer Deep Dive

  Huang et al. (2020) in their TabTransformer: Tabular Data Modeling Using Contextual Embeddings
- Linear Numerical Embeddings

  Gorishniy et al. (2021) in their Revisiting Deep Learning Models for Tabular Data
- Piecewise Linear & Periodic Encoding

  Gorishniy et al. (2022) in On Embeddings for Numerical Features in Tabular Deep Learning
- AdHoc: TabNet

  Sercan O. Arık, Tomas Pfister in TabNet: Attentive Interpretable TabNet: Attentive Interpretable Tabular Learning

Motivation

Based on the series of posts https://towardsdatascience.com/transformers-for-tabular-data-tabtransformer-deep-dive-5fb2438da820

What is tabular data?

Tabular data is organized in a table with rows and columns.

”Standard dataset type” from classic ML course.

Tabular data is ubiquitous.

Can include various data & feature types:

- *categorical features (features that can be stored based on the names or labels given to them)*
- *numerical features (feature in the numeric form)*

Tabular Data

columns = attributes for those observations

Rows = observations

Player	Minutes	Points	Rebounds	Assists
A	41	20	6	5
B	30	29	7	6
C	22	7	7	2
D	26	3	3	9
E	20	19	8	0
F	9	6	14	14
G	14	22	8	3
I	22	36	0	9
J	34	8	1	3

Classic datasets examples

California Housing Dataset (Regression problem)

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Target
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

Titanic dataset (Binary classification task)

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Motivation

Tabular domain is still mainly dominated by *gradient boosted decision trees (GBDT)*

Question: is it possible to bridge the gap?

Photo by Samule Sun on Unsplash

Motivation

Tabular domain is still mainly dominated by *gradient boosted decision trees (GBDT)*

Question: is it possible to bridge the gap?

Answer: Yes!

Huang et al. (2020) in their [TabTransformer: Tabular Data Modeling Using Contextual Embeddings]

Computer Science > Machine Learning

[Submitted on 11 Dec 2020]

TabTransformer: Tabular Data Modeling Using Contextual Embeddings

Xin Huang, Ashish Khetan, Milan Cvitkovic, Zohar Karnin

We propose TabTransformer, a novel deep tabular data modeling architecture for supervised and semi-supervised learning. The TabTra attention based Transformers. The Transformer layers transform the embeddings of categorical features into robust contextual embedd accuracy. Through extensive experiments on fifteen publicly available datasets, we show that the TabTransformer outperforms the state methods for tabular data by at least 1.0% on mean AUC, and matches the performance of tree-based ensemble models. Furthermore, we embeddings learned from TabTransformer are highly robust against both missing and noisy data features, and provide better interpreta supervised setting we develop an unsupervised pre-training procedure to learn data-driven contextual embeddings, resulting in an aver

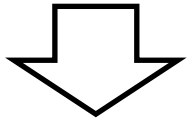
Photo by Samule Sun on Unsplash

TabTransformers

Based on the series of posts https://towardsdatascience.com/transformers-for-tabular-data-tabtransformer-deep-dive-5fb2438da820

Idea

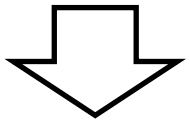
Idea: use transformers to transforms regular categorical embeddings into contextual ones.



Transformers can significantly improve in the performance of regular *Multi-layer Perceptron (MLP).*

Categorical embeddings

In DL to use categorical features we train their embeddings.

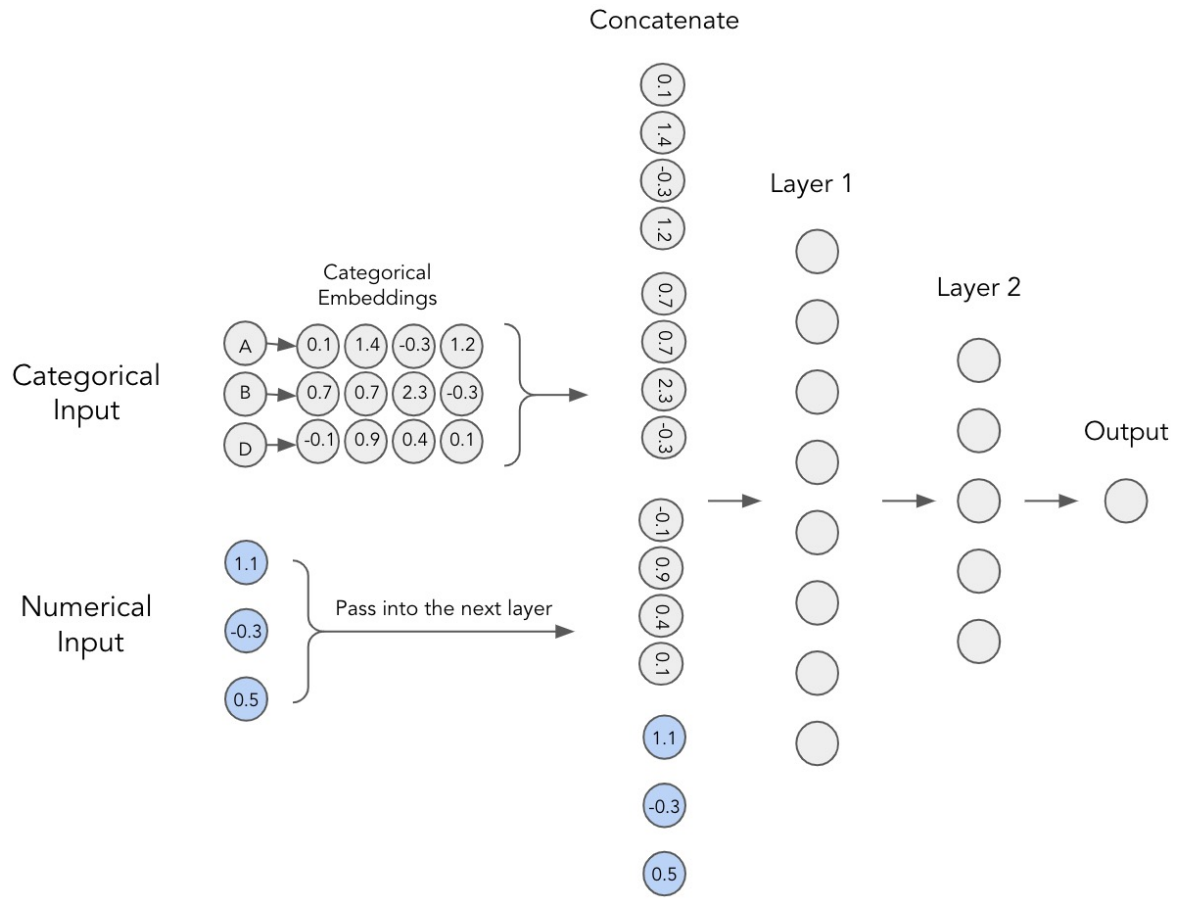


Each categorical value gets a unique dense vector representation which is passed on to the next layers.

Categorical embeddings

Example:

- *Each categorical feature gets represented as a 4D array (embedding).*
- *These embeddings are then concatenated with numerical features and are used as inputs to the MLP.*

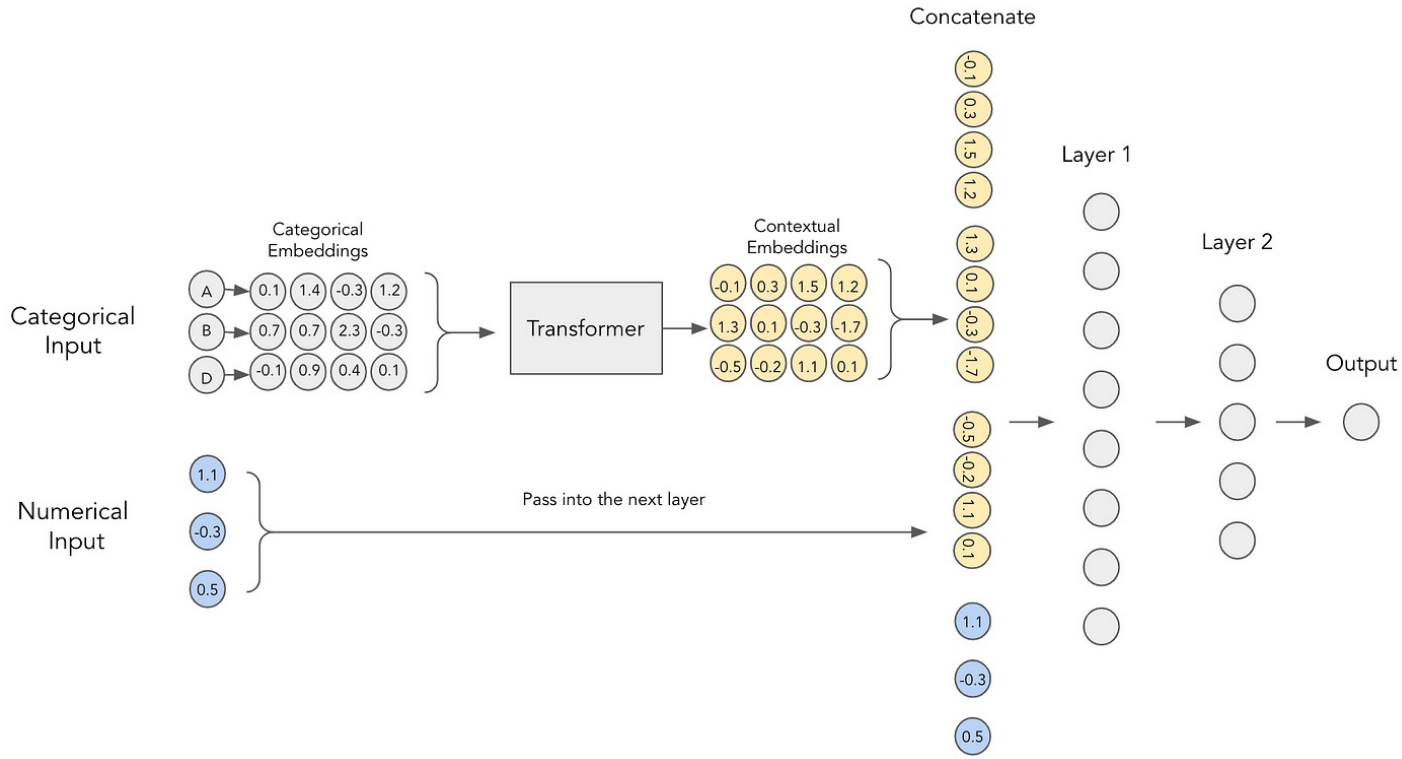


MLP with categorical embeddings.

Contextual embeddings

Problem: categorical embeddings lack the context meaning, they do not encode any interactions and relationships between the categorical variables.

Solution: use the Transformers to contextualize the embeddings

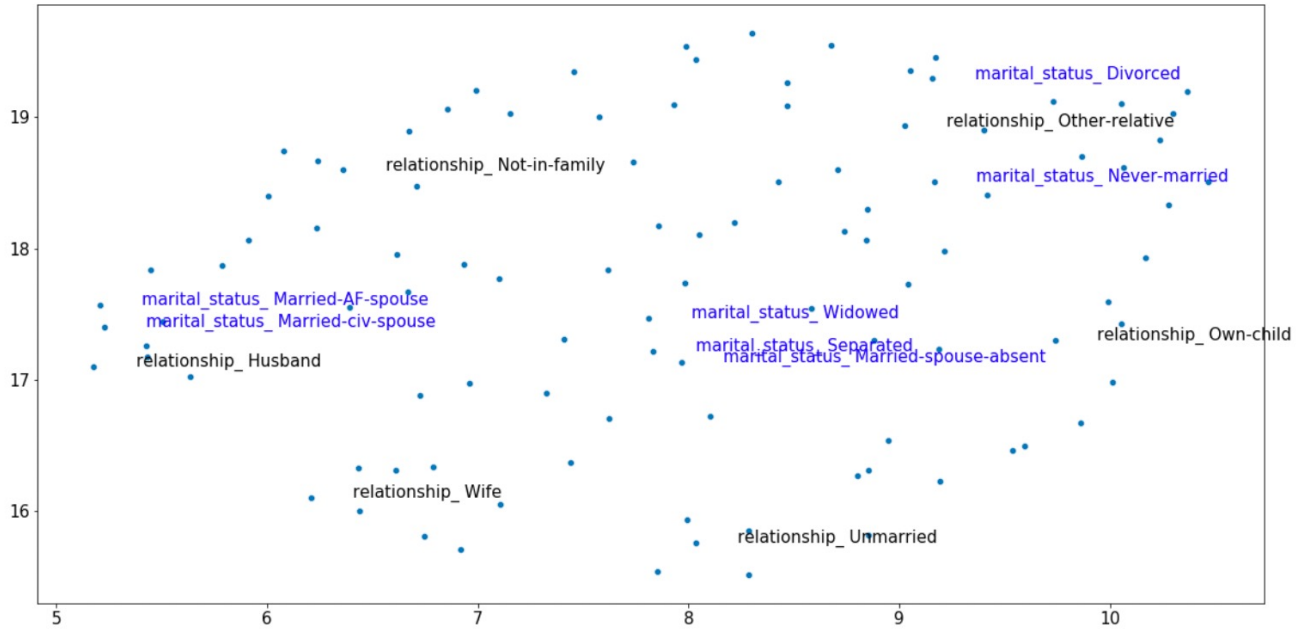


Contextual embeddings.

Contextual embeddings

Example: two categorical features — *relationship (black)* and *marital status (blue).*

Features are related => values of “Married”, “Husband” & “Wife” should be close to each other in the vector space.



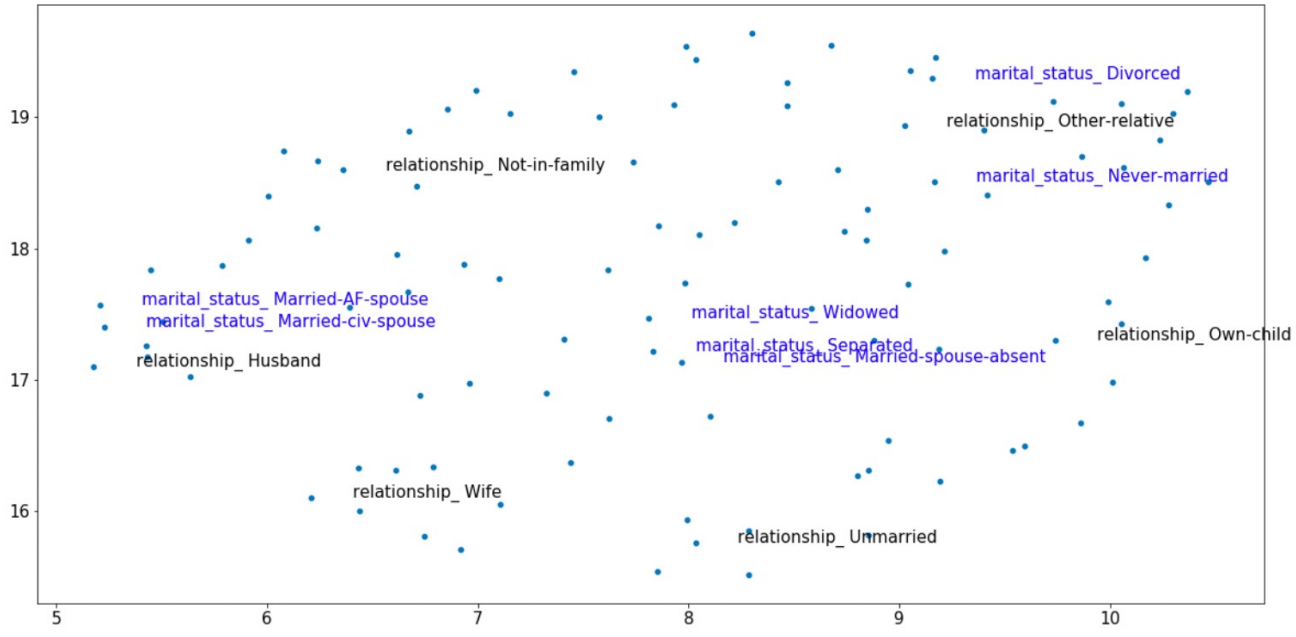
Example of trained TabTransformer Embeddings.

Contextual embeddings

Example: two categorical features — *relationship (black)* and *marital status (blue).*

Features are related => values of “Married”, “Husband” & “Wife” should be close to each other in the vector space.

With contextual embeddings “Married” is closer to “Husband” and “Wife”, whereas “non-married” categorical values from separate clusters to the right.

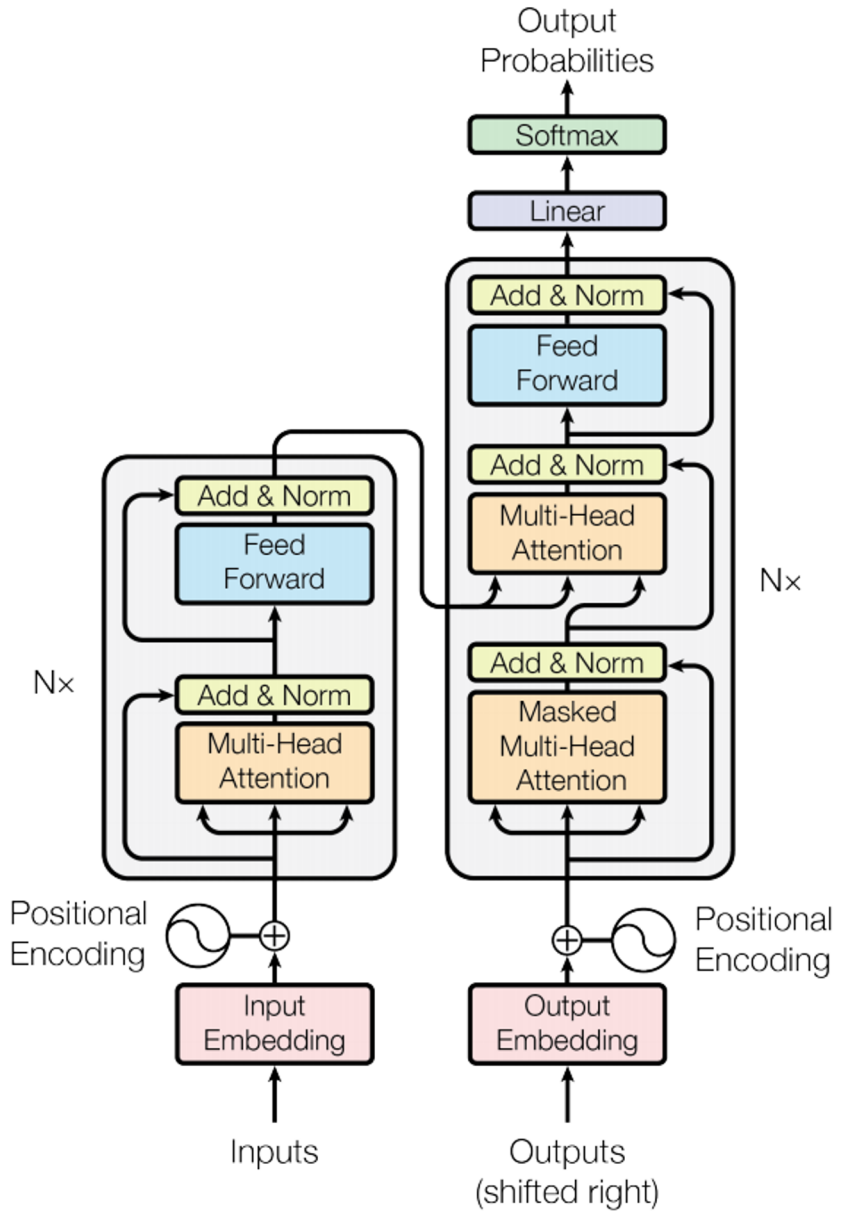


Example of trained TabTransformer Embeddings.

The transformer

The Transformer consists of encoder & decoder.

What part do we need for the contextual embeddings?

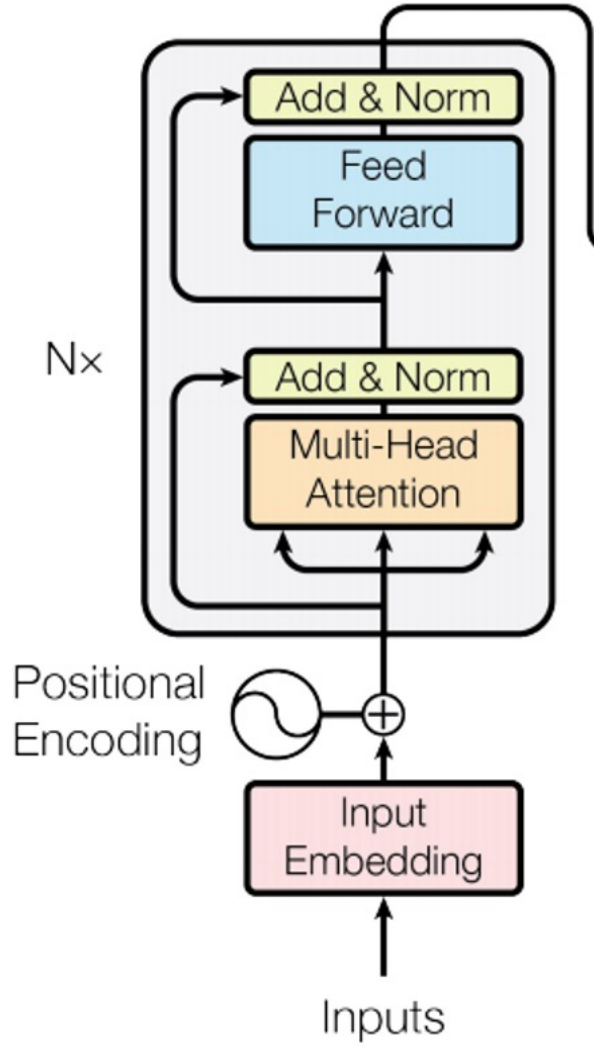


Vaswani et al, 2017. Attention is all you need.

The transformer

What part do we need for the contextual embeddings?

For the TabTransformer, we need **the encoder** for contextualizing the input embeddings.



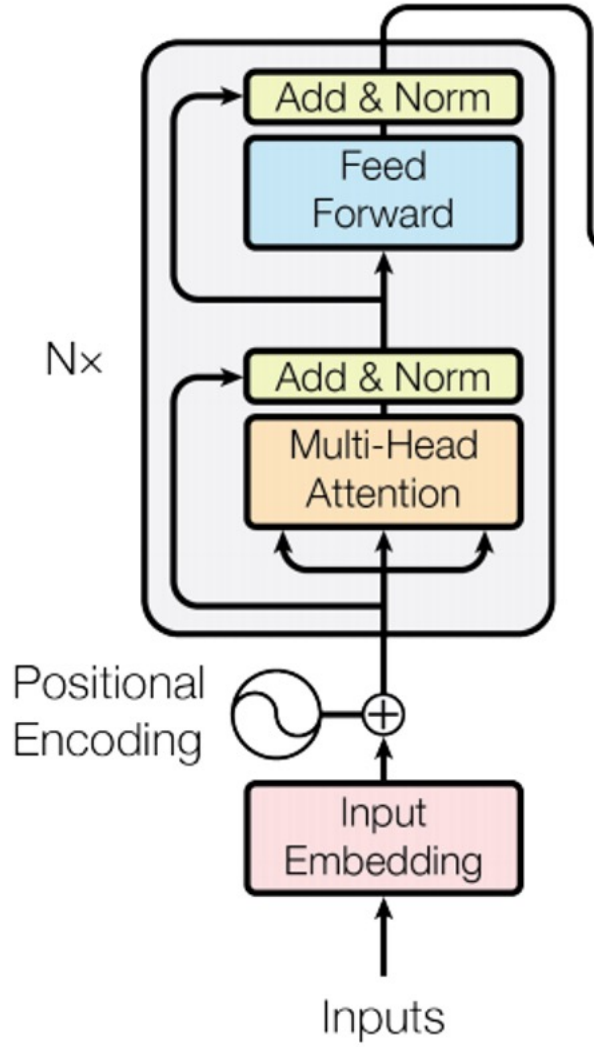
Vaswani et al, 2017. Attention is all you need.

The transformer

What part do we need for the contextual embeddings?

Encoder:

1. *Embedding inputs*
2. *Multi-head attention*
3. *Skip-connection & normalization*
4. *Feed-Forward MLP*
5. *Skip-connection & normalization*



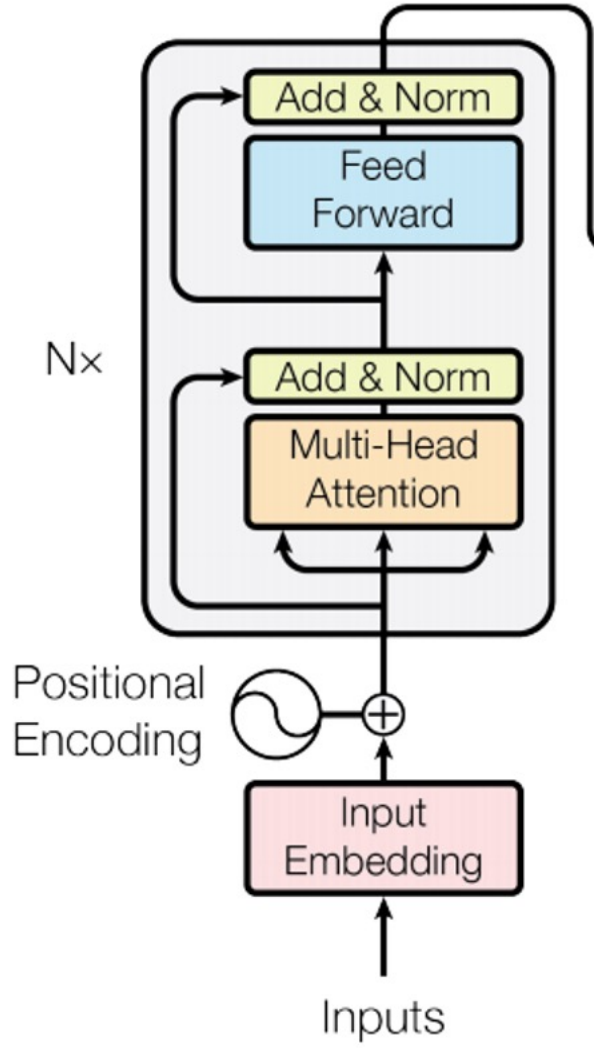
Vaswani et al, 2017. Attention is all you need.

The transformer

What part do we need for the contextual embeddings?

For the TabTransformer, we need **the encoder** for *contextualizing the input embeddings*.

But how?



Vaswani et al, 2017. Attention is all you need.

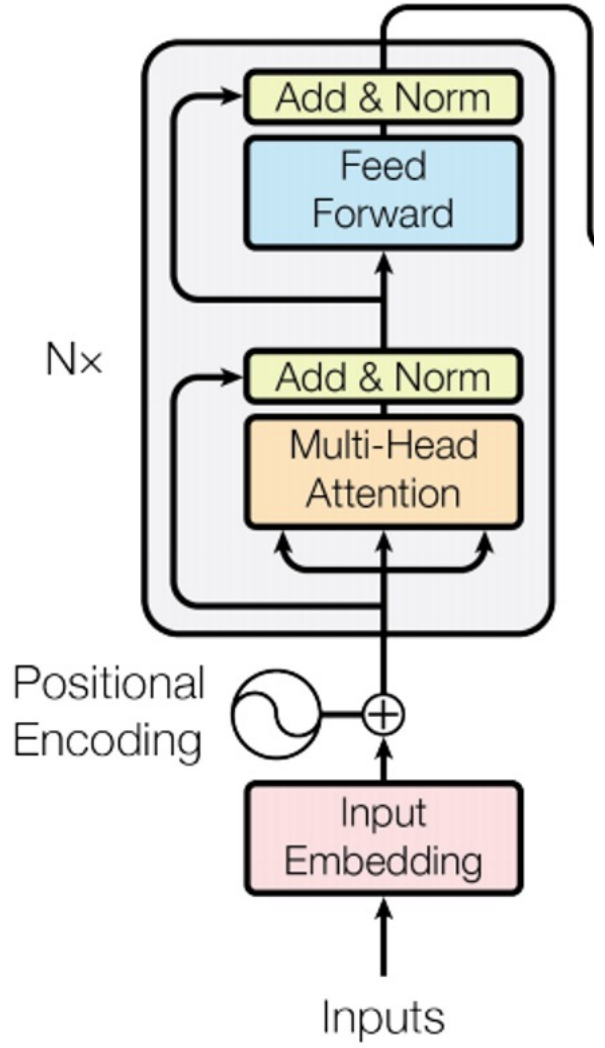
The transformer

What part do we need for the contextual embeddings?

For the TabTransformer, we need **the encoder** for *contextualizing the input embeddings*.

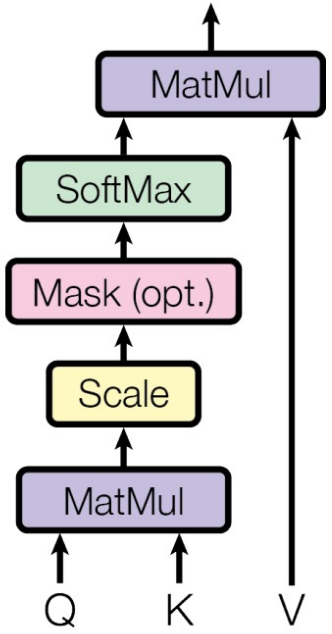
But how?

Answer: multi-head attention mechanism**!**



Vaswani et al, 2017. Attention is all you need.

Scaled dot-product attention



The key concept behind self attention is that it allows the network to ***learn how best to route information between pieces of a an input sequence.***

It helps the model **to figure out which parts of the input are more important** and which ones are less when **representing a certain category** (word).

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

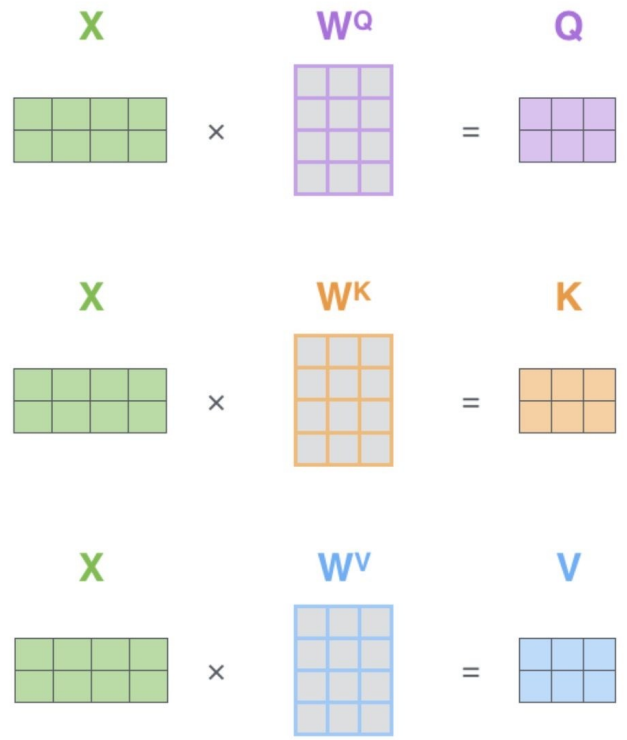
Rush. The Annotated Transformer. https://nlp.seas.harvard.edu/2018/04/03/attention.html

For words

Query, Key and Value vectors

Each **token embedding** gets three representations:

- **query** - asking for information;
- **key** - saying that it has some information;
- **value** - giving the information.



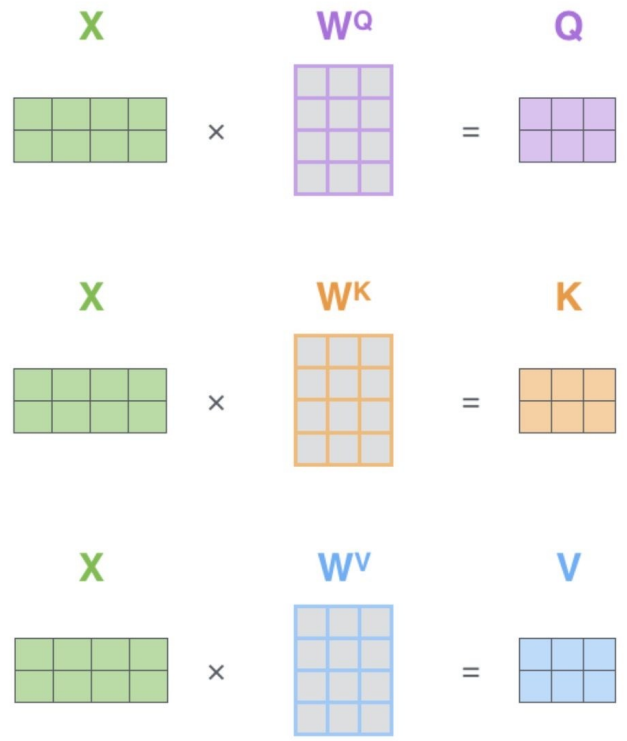
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}})\mathbf{V}$$

For categorical data

Query, Key and Value vectors

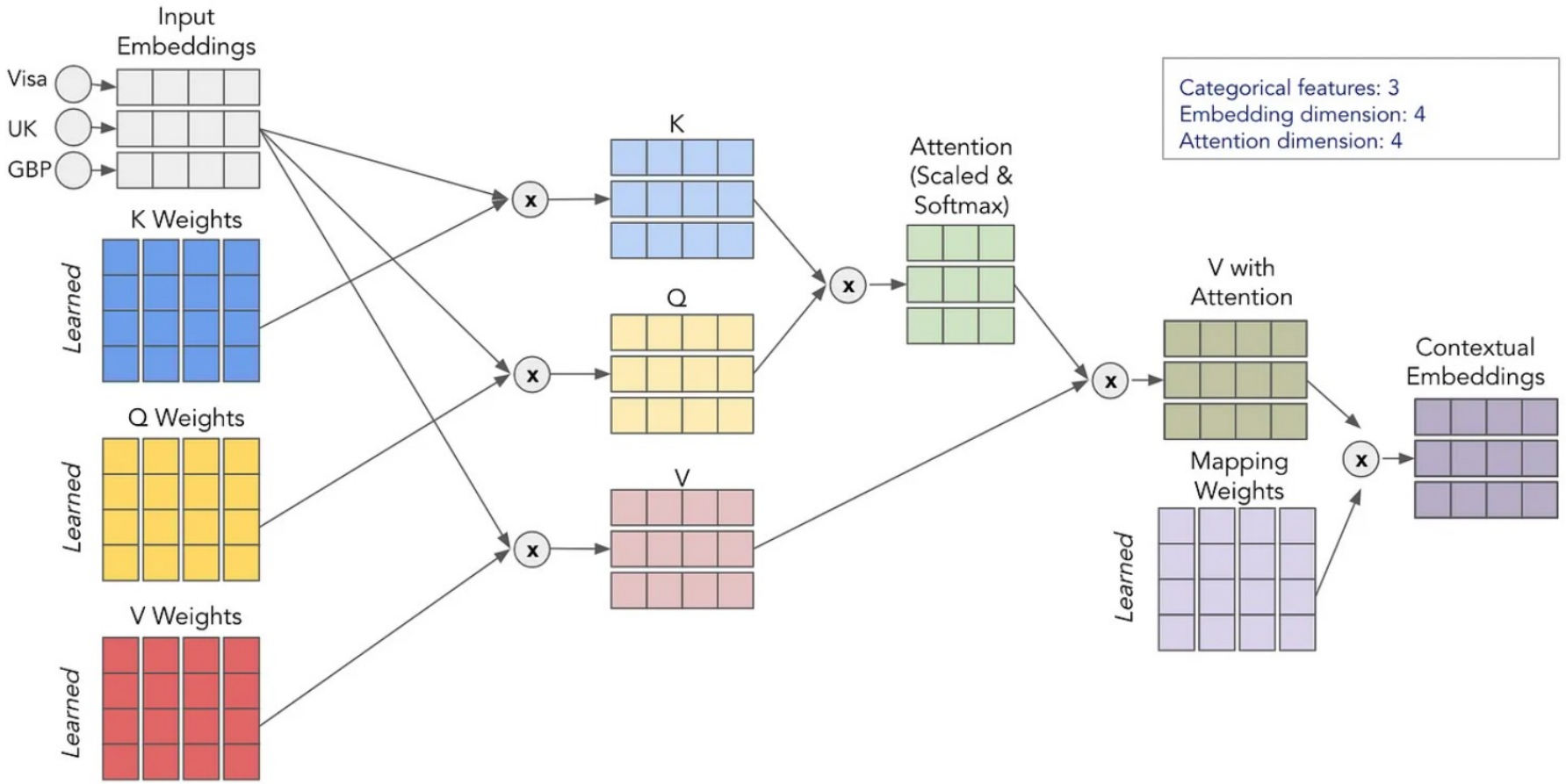
Each **categorical embeddings for each categorical feature** gets three representations:

- **query** - asking for information;
- **key** - saying that it has some information;
- **value** - giving the information.



$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}})\mathbf{V}$$

For categorical data



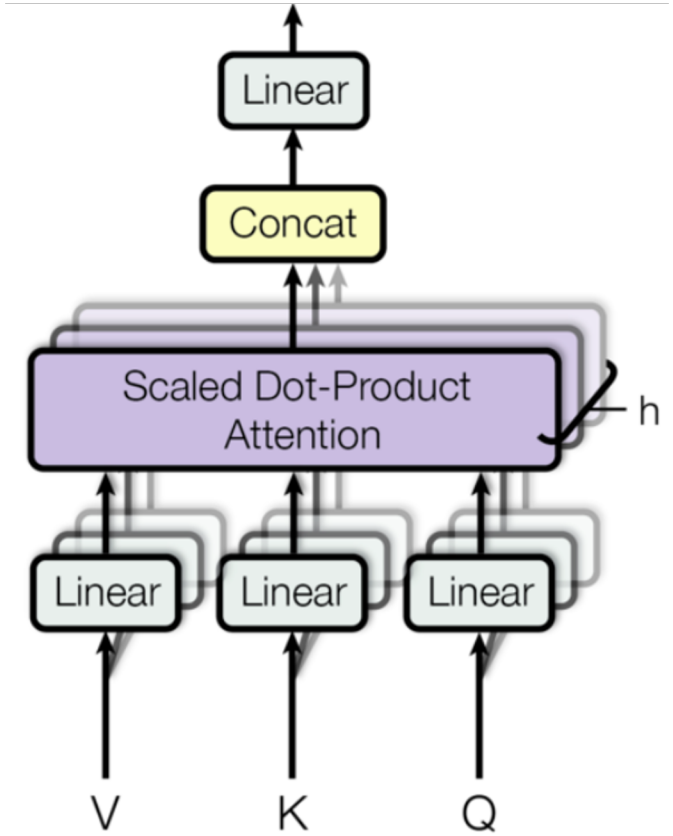
Attention flow visualized.

Multi-headed attention

Create *h* heads (with different Q, K, V matrices)

⇩

Multiple contextual embeddings which form our final Multi-Head Attention



Vaswani et al, 2017. Attention is all you need.

Altogether

- Categorical embeddings do not include contextual information

Altogether

- Categorical embeddings do not include contextual information
- **Contextualize the embeddings** by passing categorical embeddings through Transformer Encoder

Altogether

- Categorical embeddings do not include contextual information
- **Contextualize the embeddings** by passing categorical embeddings through Transformer Encoder
- Contextualize embeddings thanks to **Multi-Head Attention**

Altogether

- Categorical embeddings do not include contextual information
- **Contextualize the embeddings** by passing categorical embeddings through Transformer Encoder
- Contextualize embeddings thanks to **Multi-Head Attention**
- Use matrices Q, K and V to **find useful interactions and correlations** while encoding the variables

Altogether

- Categorical embeddings do not include contextual information
- **Contextualize the embeddings** by passing categorical embeddings through Transformer Encoder
- Contextualize embeddings thanks to **Multi-Head Attention**
- Use matrices Q, K and V to **find useful interactions and correlations** while encoding the variables
- In *TabTransformer*, contextualized embeddings are *concatenated with numerical input and passed through a simple MLP* to output a prediction

Results

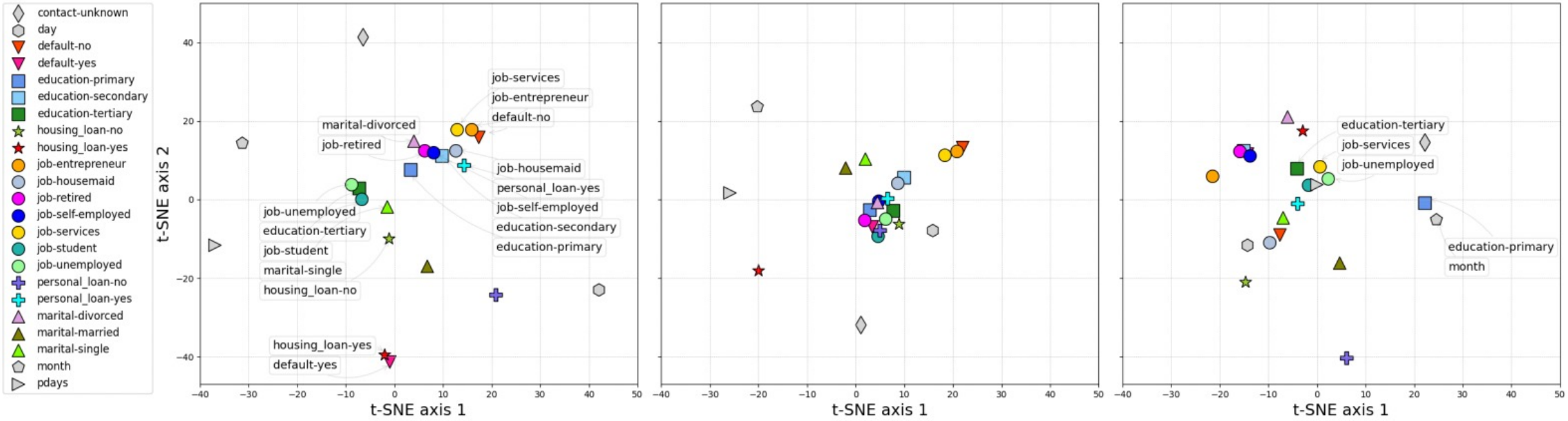
Evaluation on 15 tabular datasets

Dataset	Baseline MLP	TabTransformer	Gain (%)
albert	74.0	75.7	**1.7**
1995_income	90.5	90.6	**0.1**
dota2games	63.1	63.3	**0.2**
hcdr_main	74.3	75.1	**0.8**
adult	72.5	73.7	**1.2**
bank_marketing	92.9	93.4	**0.5**
blastchar	83.9	83.5	-0.4
insurance_co	69.7	74.4	**4.7**
jasmine	85.1	85.3	**0.2**
online_shoppers	91.9	92.7	**0.8**
philippine	82.1	83.4	**1.3**
qsar_bio	91.0	91.8	**0.8**
seismicbumps	73.5	75.1	**1.6**
shrutime	84.6	85.6	**1.0**
spambase	98.4	98.5	**0.1**

Comparison between TabTransfomers and the baseline MLP. The evaluation metric is AUC in percentage. Adapted from Huang et al. (2020)

Results

t-SNE plots of learned embeddings for categorical features on dataset BankMarketing.
Left: TabTransformer-the embeddings generated from the last layer of the attention-based Transformer.
Center: TabTransformer-the embeddings before being passed into the attention-based Transformer**.**
Right: The embeddings learned from MLP.



Adapted from [Huang et al. (2020)]

Results

# Labeled data	50	200	500
TabTransformer-RTD	78.6 ± 0.6	$\mathbf{81.6} \pm 0.5$	$\mathbf{83.4} \pm 0.5$
TabTransformer-MLM	78.5 ± 0.6	81.0 ± 0.6	82.4 ± 0.5
MLP (ER)	$\mathbf{79.4} \pm 0.6$	81.1 ± 0.6	82.3 ± 0.6
MLP (PL)	79.1 ± 0.6	81.1 ± 0.6	82.0 ± 0.6
TabTransformer (ER)	77.9 ± 0.6	81.2 ± 0.6	82.1 ± 0.6
TabTransformer (PL)	77.8 ± 0.6	81.0 ± 0.6	82.1 ± 0.6
MLP (DAE)	78.5 ± 0.7	80.7 ± 0.6	82.2 ± 0.6
GBDT (PL)	73.4 ± 0.7	78.8 ± 0.6	81.3 ± 0.6

Semi-supervised learning results for 12 datasets each with less than 30K data points, for different number of labeled data points. Evaluation metrics are mean AUC in percentage. Larger the number, better the result.

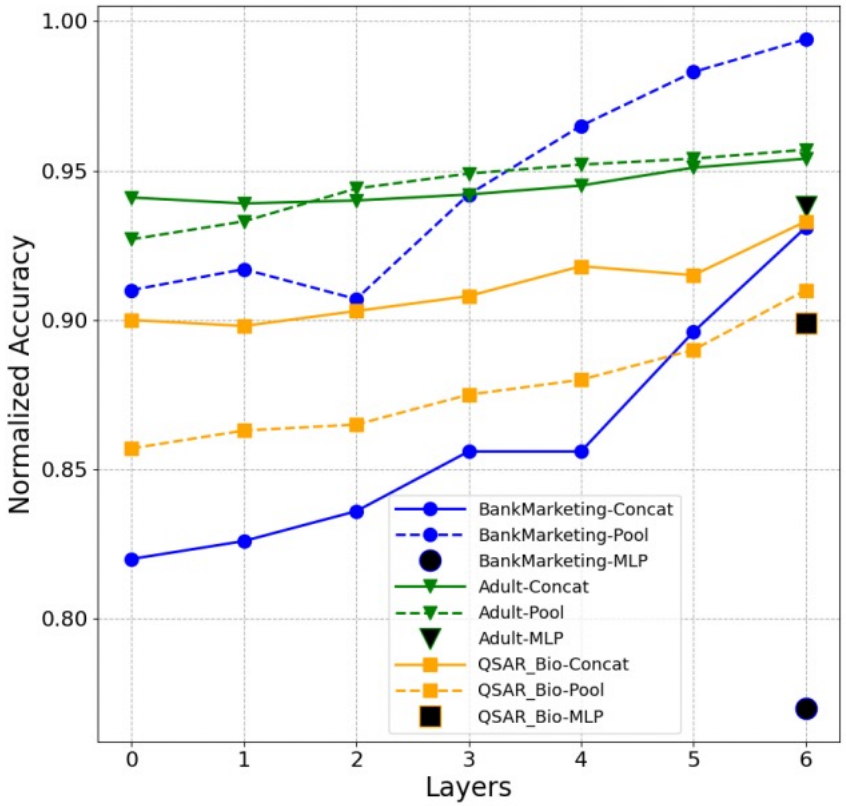
Model Name	Mean AUC (%)
TabTransformer	$\mathbf{82.8} \pm 0.4$
MLP	81.8 ± 0.4
GBDT	$\mathbf{82.9} \pm 0.4$
Sparse MLP	81.4 ± 0.4
Logistic Regression	80.4 ± 0.4
TabNet	77.1 ± 0.5
VIB	80.5 ± 0.4

Model performance in supervised learning. The evaluation metric is mean ± standard deviation of AUC score over the 15 datasets for each model. Larger the number, better the result. The top 2 numbers are bold.

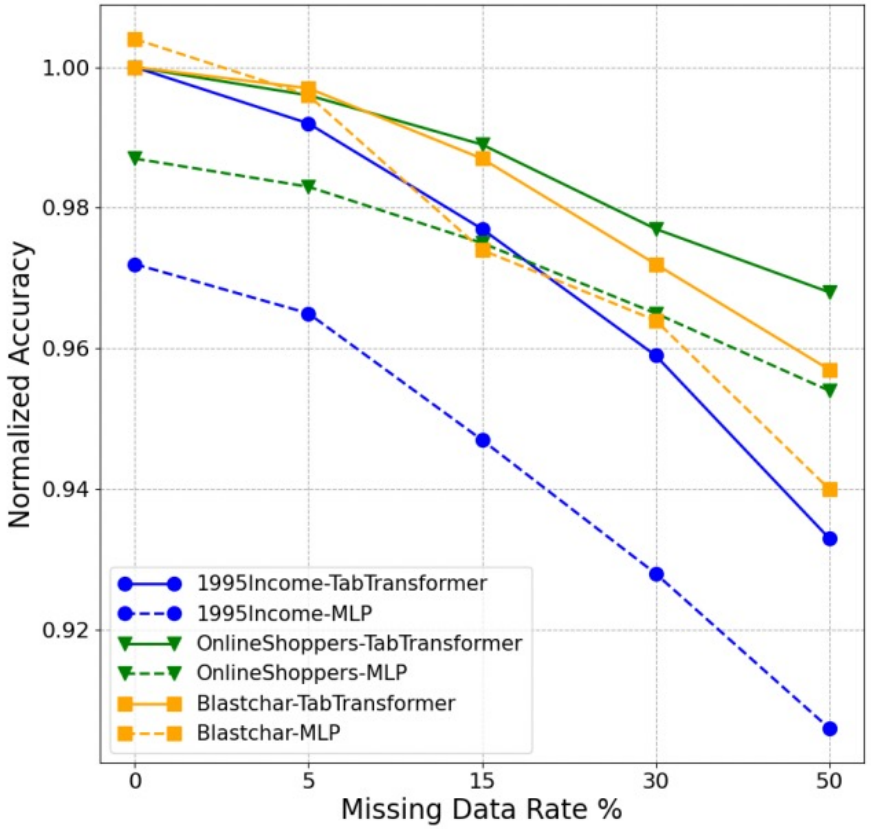
Adapted from Huang et al. (2020)

Results

Performance of TabTransformer and MLP with noisy data. For each dataset, each prediction score is normalized by the score of TabTransformer at 0 noise.



Performance of TabTransformer and MLP under missing data scenario. For each dataset, each prediction score is normalized by the score of TabTransformer trained without missing values.



Results. Adapted from [Huang et al. (2020)]

Linear Numerical Embeddings

Based on the series of posts https://towardsdatascience.com/improving-tabtransformer-part-1-linear-numerical-embeddings-dbc3be3b5bb5

Motivation

TabTransformer does wonders for categorical data.

Question: But what about contextualizing numerical features?

Photo by Samule Sun on Unsplash

Motivation

TabTransformer does wonders for categorical data.

Question: But what about contextualizing numerical features?

Idea: apply Transformer for the numerical features as well!

Photo by Samule Sun on Unsplash

Motivation

TabTransformer does wonders for categorical data.

Question: But what about contextualizing numerical features?

Idea: apply Transformer for the numerical features as well!

Gorishniy et al. (2021) in their Revisiting Deep Learning Models for Tabular Data

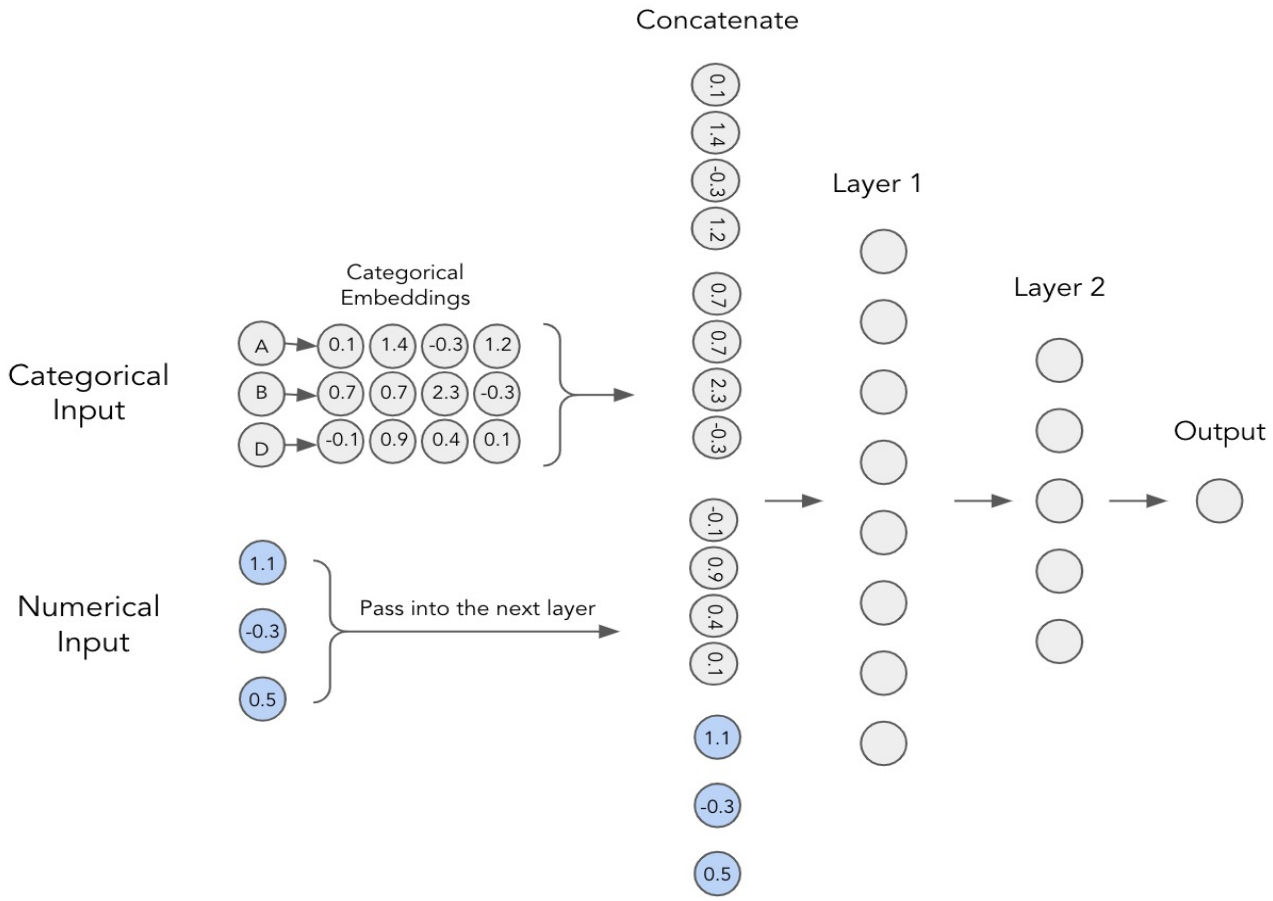
Photo by Samule Sun on Unsplash

Revisiting Deep Learning Models for Tabular Data

Motivation

In vanilla TabTransformer we do not encode numerical features, we simply concatenate them with contextualized categorical embeddings

What a pitty =(



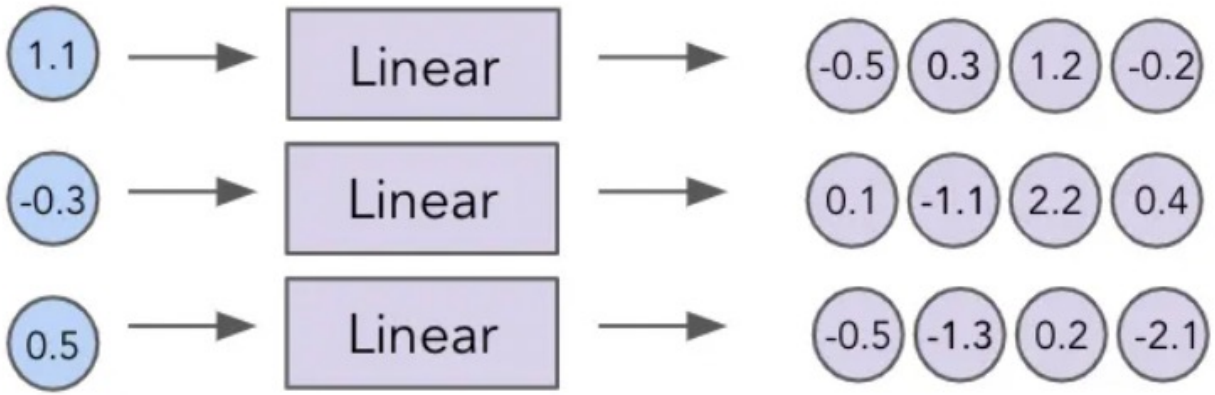
MLP with categorical embeddings.

Idea

In vanilla TabTransformer we do not encode numerical features, we simply concatenate them with contextualized categorical embeddings

What a pitty =(

Idea: let's embed numerical features as well!

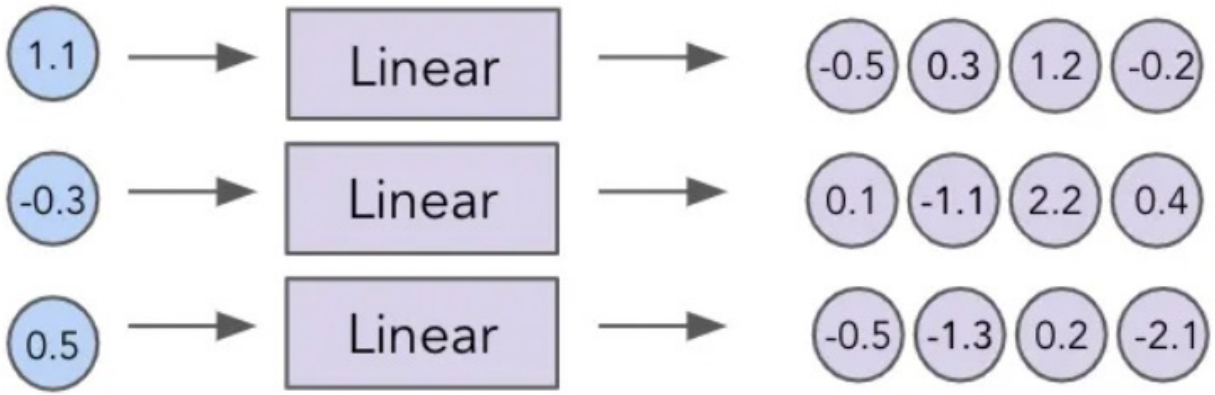


Linear Numerical Embeddings.

Idea

Idea: let's embed numerical features as well!

Each feature is transformed into a dense vector after passing through *a simple fully connected layer.*



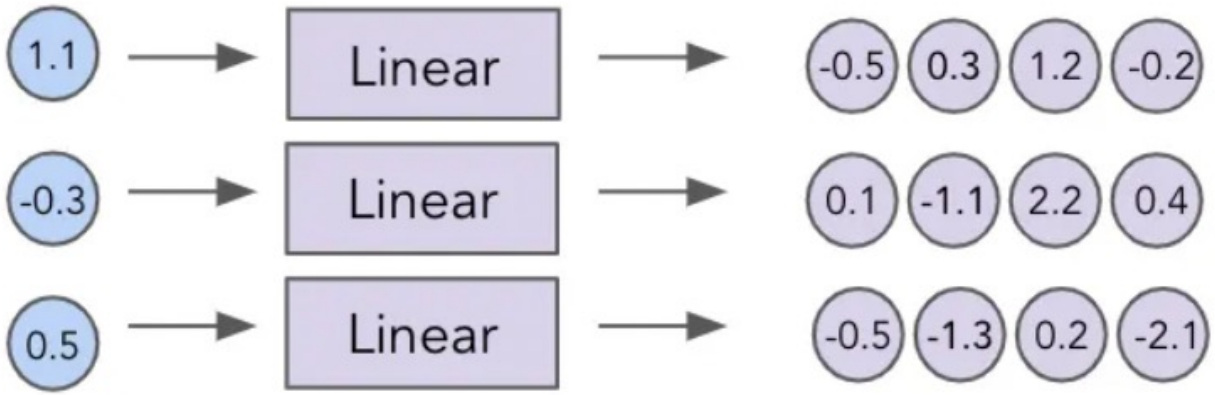
Linear Numerical Embeddings.

Method

Idea: let's embed numerical features as well!

Each feature is transformed into a dense vector after passing through *a simple fully connected layer.*

Question: why would we do that if these features are already numeric?

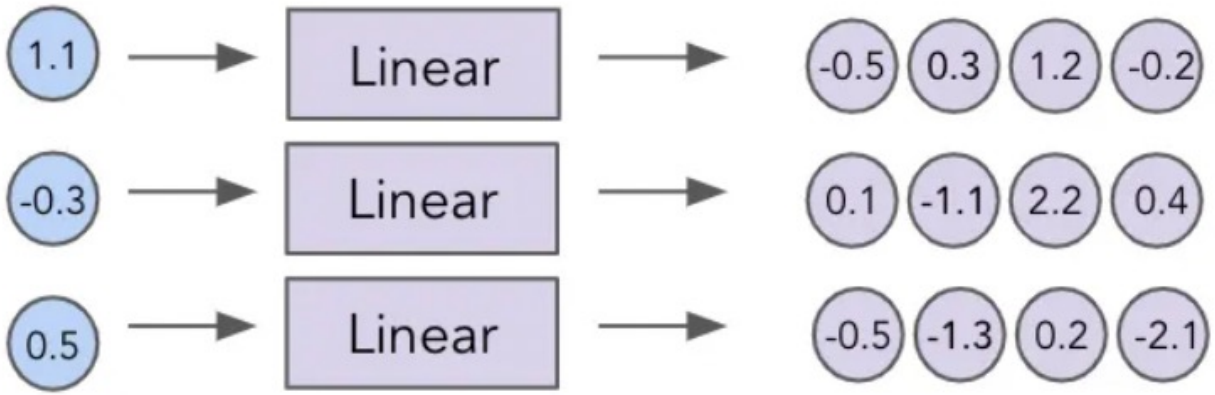


Linear Numerical Embeddings.

Method

Question: why would we do that if these features are already numeric?

Result: now we can pass numerical embeddings through the Transformer blocks as well!



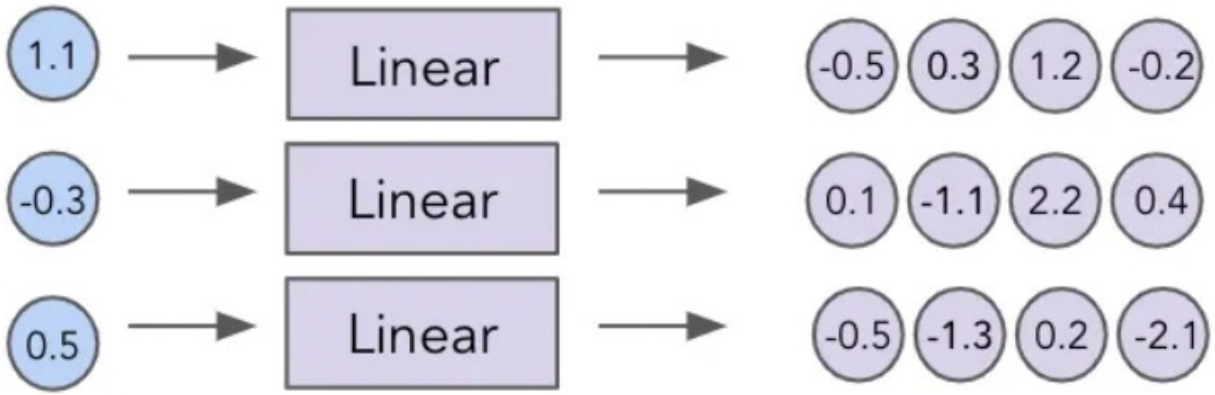
Linear Numerical Embeddings.

Method

Result: now we can pass numerical embeddings through the Transformer blocks as well!

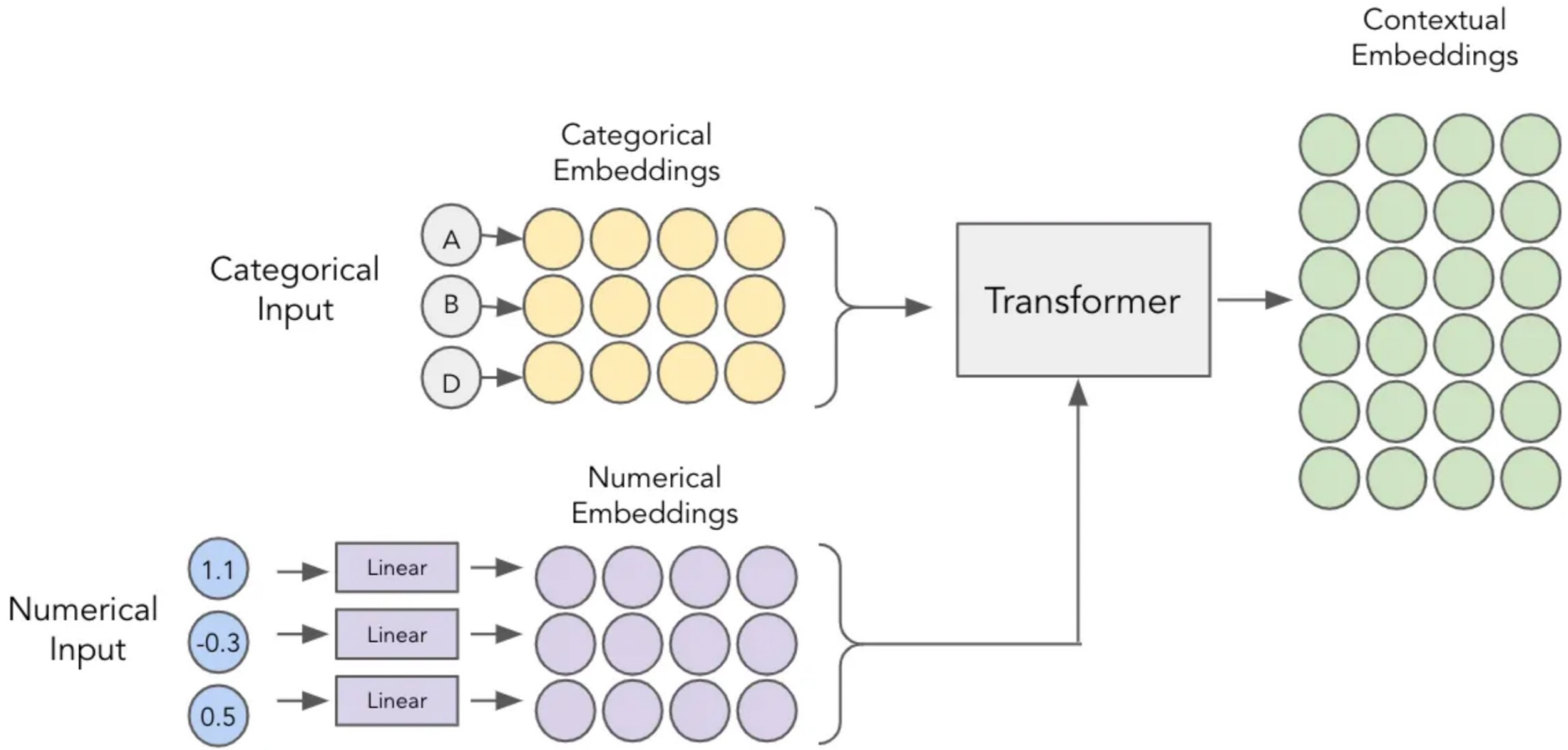
Add more context to learn from.

Improve the representation quality.



Linear Numerical Embeddings.

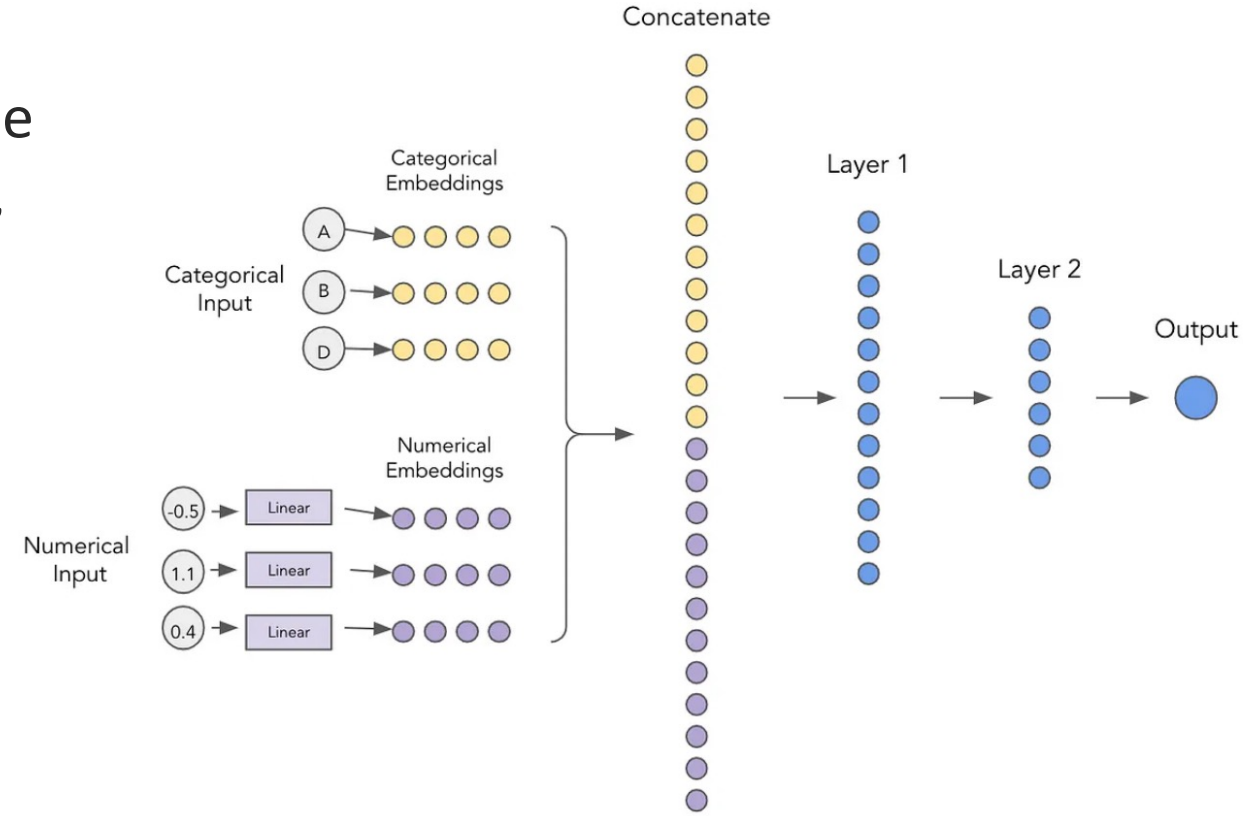
Method



Transformer with Numerical Embeddings.

Linear embeddings

Numerical embeddings can be used with various DL models, even simple MPLs, not only TabTransformer.



MLP with Numerical Embeddings.

Linear embeddings

	GE ↑	CH ↑	EY ↑	CA ↓	HO ↓	AD ↑	OT ↑	HI ↑	FB ↓	SA ↑	CO ↑	MI ↓
MLP	0.632	0.856	0.615	0.495	3.204	0.854	**0.818**	0.720	5.686	0.912	0.964	0.747
MLP-P	0.631	**0.860**	0.701	0.489	3.129	0.869	0.807	0.723	5.845	0.923	0.968	0.747
MLP-PL	0.641	**0.859**	0.866	**0.467**	3.113	0.868	**0.819**	0.727	**5.530**	0.924	**0.969**	0.746
MLP-PLR	**0.674**	**0.857**	**0.920**	**0.467**	**3.050**	**0.870**	**0.819**	**0.728**	**5.525**	**0.924**	**0.970**	**0.746**
Transformer-L	**0.632**	0.860	0.731	**0.465**	3.239	0.858	**0.817**	0.725	**5.602**	**0.924**	**0.971**	**0.746**
Transformer-PLR	**0.646**	**0.863**	**0.940**	**0.464**	**3.162**	**0.870**	0.814	**0.730**	5.760	**0.924**	**0.972**	**0.746**

L – Linear
LR – ReLU ◦ Linear
P – Periodic
PL – Linear ◦ Periodic
PLR – ReLU ◦ Linear ◦ Periodic

$$f_i(x) = \texttt{Periodic}(x) = \texttt{concat}[\sin(v),\ \cos(v)]$$
$$v = [2\pi c_1 x,\ \ldots,\ 2\pi c_k x]$$

Yury Gorishniy et al, On Embeddings for Numerical Features in Tabular Deep Learning

CLS token

Adapt the idea of CLS token from NLP

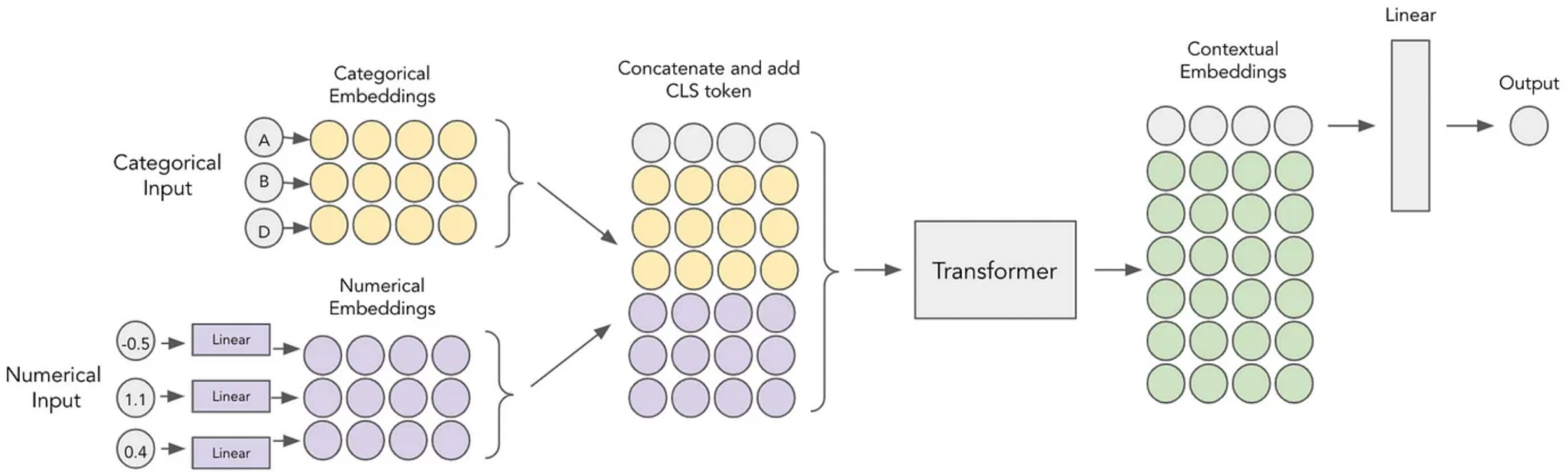
Recap question: how do we use CLS token in NLP?

CLS token

Adapt the idea of CLS token from NLP:

- append CLS embedding after embedding the features
- contextualize categorical, numerical and CLS embeddings by passing them through the Transformer
- use contextualized CLS token embedding as an input into a simple MLP classifier which produces the desired output

Altogether



FT-Transformer.

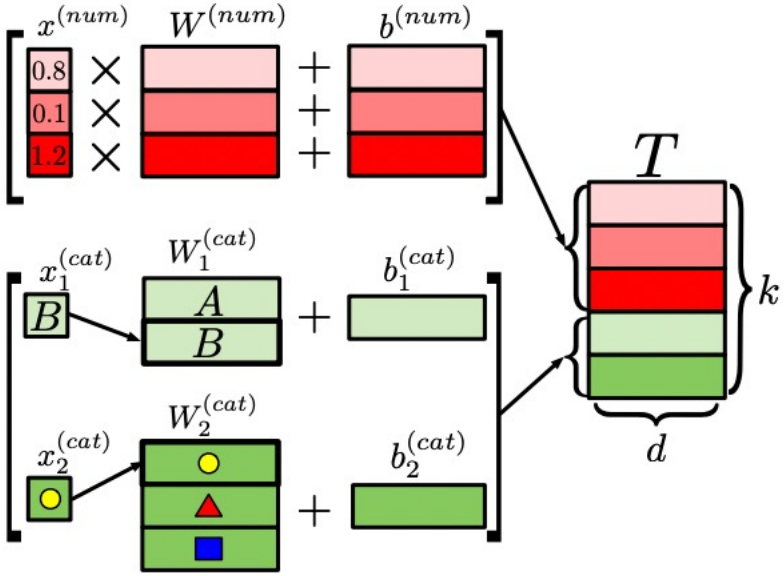
Mathematics

Feature Tokenizer

$$T_j^{(num)} = b_j^{(num)} + x_j^{(num)} \cdot W_j^{(num)} \in \mathbb{R}^d,$$

$$T_j^{(cat)} = b_j^{(cat)} + e_j^T W_j^{(cat)} \in \mathbb{R}^d,$$

$$T = \texttt{stack}\left[T_1^{(num)}, \ldots, T_{k^{(num)}}^{(num)}, T_1^{(cat)}, \ldots, T_{k^{(cat)}}^{(cat)}\right] \in \mathbb{R}^{k \times d}.$$



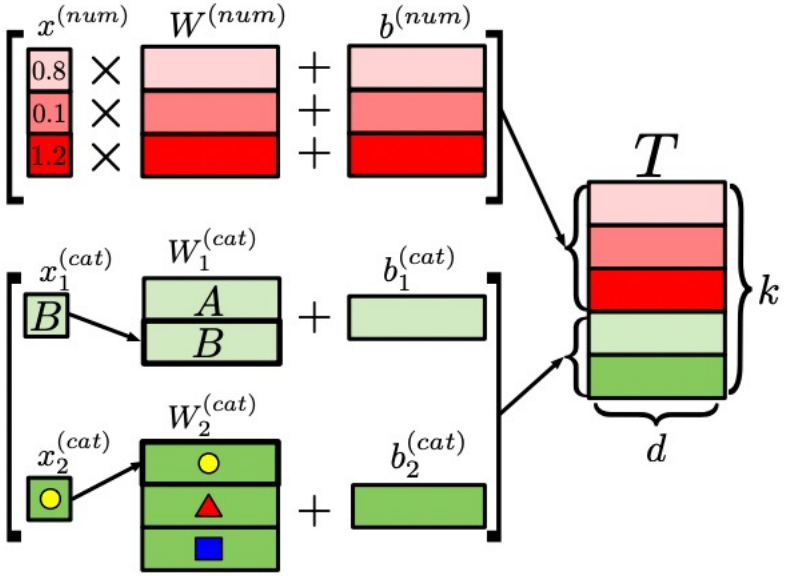
Feature Tokenizer; in the example, there are three numerical and two categorical features.

Gorishniy et al. (2021)

Mathematics

Feature Tokenizer

$$T_j^{(num)} = b_j^{(num)} + x_j^{(num)} \cdot W_j^{(num)} \in \mathbb{R}^d,$$
$$T_j^{(cat)} = b_j^{(cat)} + e_j^T W_j^{(cat)} \in \mathbb{R}^d,$$
$$T = \texttt{stack}\left[T_1^{(num)}, \ldots, T_{k^{(num)}}^{(num)}, T_1^{(cat)}, \ldots, T_{k^{(cat)}}^{(cat)}\right] \in \mathbb{R}^{k \times d}.$$

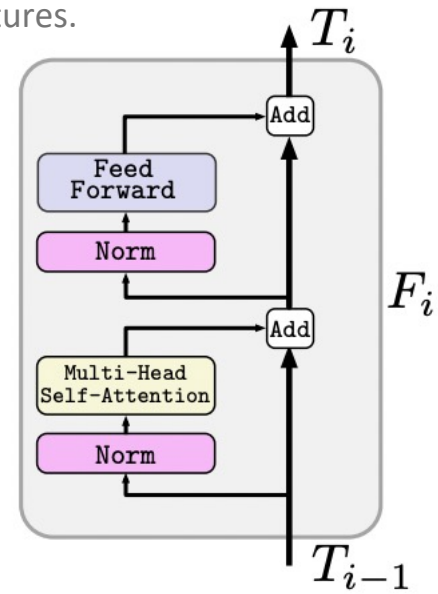


Feature Tokenizer; in the example, there are three numerical and two categorical features.

Transformer

$$T_0 = \texttt{stack}\left[\texttt{[CLS]}, T\right] \qquad T_i = F_i(T_{i-1}).$$

$F_i - i_{th}$ transformer layer



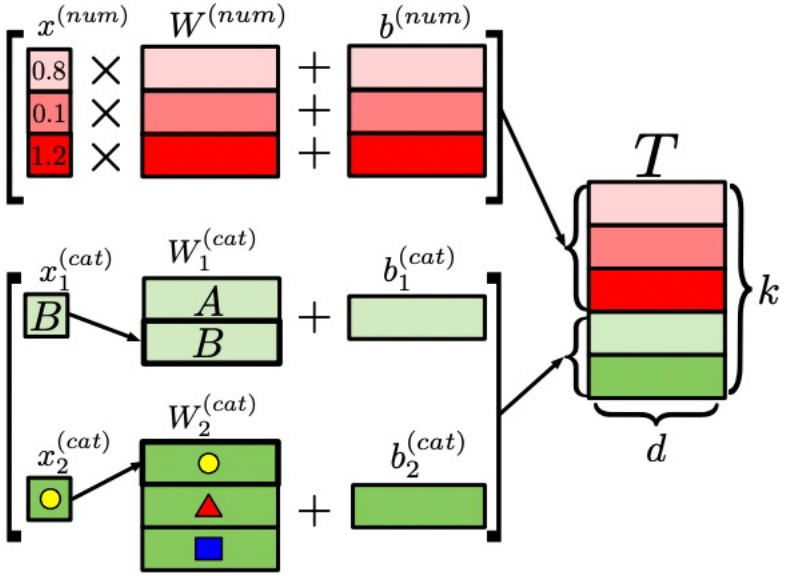
One Transformer layer.

Gorishniy et al. (2021)

Mathematics

Feature Tokenizer

$$T_j^{(num)} = b_j^{(num)} + x_j^{(num)} \cdot W_j^{(num)} \in \mathbb{R}^d,$$
$$T_j^{(cat)} = b_j^{(cat)} + e_j^T W_j^{(cat)} \in \mathbb{R}^d,$$
$$T = \texttt{stack}\left[T_1^{(num)}, \ldots, T_{k^{(num)}}^{(num)}, T_1^{(cat)}, \ldots, T_{k^{(cat)}}^{(cat)}\right] \in \mathbb{R}^{k \times d}.$$

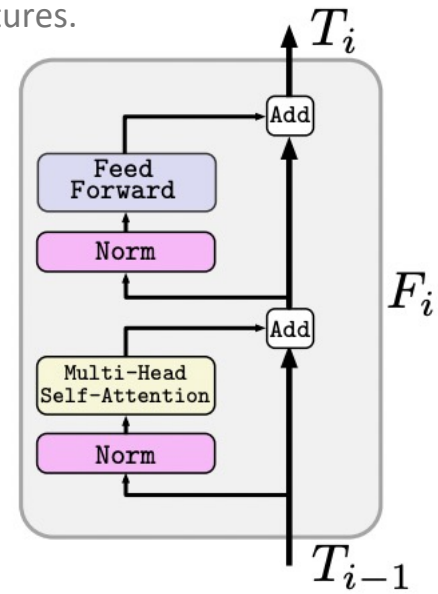


Feature Tokenizer; in the example, there are three numerical and two categorical features.

Transformer

$$T_0 = \texttt{stack}\left[\texttt{[CLS]}, T\right] \qquad T_i = F_i(T_{i-1}).$$

$F_i - i_{th}$ transformer layer



One Transformer layer.

Prediction

For the final prediction final representation of the [CLS] token is used.

$$\hat{y} = \texttt{Linear}(\texttt{ReLU}(\texttt{LayerNorm}(T_L^{\texttt{[CLS]}})))$$

Gorishniy et al. (2021)

Results

Comparison with DL models

- **SNN** (Klambauer et al., 2017). An MLP-like architecture with the SELU activation that enables training deeper models.
- **NODE** (Popov et al., 2020). A differentiable ensemble of oblivious decision trees.
- **TabNet** (Arik and Pfister, 2020). A recurrent architecture that alternates dynamical reweighing of features and conventional feed-forward modules.
- **GrowNet** (Badirli et al., 2020). Gradient boosted weak MLPs. The official implementation supports only classification and regression problems.
- **DCN V2** (Wang et al., 2020a). Consists of an MLP-like module and the feature crossing module (a combination of linear layers and multiplications).
- **AutoInt** (Song et al., 2019). Transforms features to embeddings and applies a series of attention-based transformations to the embeddings.

Table 2: Results for DL models. The metric values averaged over 15 random seeds are reported. See supplementary for standard deviations. For each dataset, top results are in **bold**. "Top" means "the gap between this result and the result with the best score is not statistically significant". For each dataset, ranks are calculated by sorting the reported scores; the "rank" column reports the average rank across all datasets. Notation: FT-T ~ FT-Transformer, ↓ ~ RMSE, ↑ ~ accuracy

	CA ↓	AD ↑	HE ↑	JA ↑	HI ↑	AL ↑	EP ↑	YE ↓	CO ↑	YA ↓	MI ↓	rank (std)
TabNet	0.510	0.850	0.378	0.723	0.719	0.954	0.8896	8.909	0.957	0.823	0.751	7.5 (2.0)
SNN	0.493	0.854	0.373	0.719	0.722	0.954	0.8975	8.895	0.961	0.761	0.751	6.4 (1.4)
AutoInt	0.474	**0.859**	0.372	0.721	0.725	0.945	0.8949	8.882	0.934	0.768	0.750	5.7 (2.3)
GrowNet	0.487	**0.857**	–	–	0.722	–	0.8970	8.827	–	0.765	0.751	5.7 (2.2)
MLP	0.499	0.852	0.383	0.719	0.723	0.954	0.8977	8.853	0.962	0.757	0.747	4.8 (1.9)
DCN2	0.484	0.853	0.385	0.716	0.723	0.955	0.8977	8.890	0.965	0.757	0.749	4.7 (2.0)
NODE	0.464	**0.858**	0.359	0.727	0.726	0.918	0.8958	**8.784**	0.958	**0.753**	**0.745**	3.9 (2.8)
ResNet	0.486	0.854	**0.396**	0.728	0.727	**0.963**	0.8969	8.846	0.964	0.757	0.748	3.3 (1.8)
FT-T	**0.459**	**0.859**	0.391	**0.732**	**0.729**	0.960	**0.8982**	8.855	**0.970**	0.756	0.746	1.8 (1.2)

Gorishniy et al. (2021)

Results

Comparison with DL models takeaways:

- MLP is still a good sanity check
- ResNet turns out to be an effective baseline that none of the competitors can consistently outperform.
- FT-Transformer performs best on most tasks and becomes a new powerful solution for the field.
- Tuning makes simple models such as MLP and ResNet competitive.

	CA ↓	AD ↑	HE ↑	JA ↑	HI ↑	AL ↑	EP ↑	YE ↓	CO ↑	YA ↓	MI ↓	rank (std)
TabNet	0.510	0.850	0.378	0.723	0.719	0.954	0.8896	8.909	0.957	0.823	0.751	7.5 (2.0)
SNN	0.493	0.854	0.373	0.719	0.722	0.954	0.8975	8.895	0.961	0.761	0.751	6.4 (1.4)
AutoInt	0.474	**0.859**	0.372	0.721	0.725	0.945	0.8949	8.882	0.934	0.768	0.750	5.7 (2.3)
GrowNet	0.487	**0.857**	–	–	0.722	–	0.8970	8.827	–	0.765	0.751	5.7 (2.2)
MLP	0.499	0.852	0.383	0.719	0.723	0.954	0.8977	8.853	0.962	0.757	0.747	4.8 (1.9)
DCN2	0.484	0.853	0.385	0.716	0.723	0.955	0.8977	8.890	0.965	0.757	0.749	4.7 (2.0)
NODE	0.464	**0.858**	0.359	0.727	0.726	0.918	0.8958	**8.784**	0.958	**0.753**	**0.745**	3.9 (2.8)
ResNet	0.486	0.854	**0.396**	0.728	0.727	**0.963**	0.8969	8.846	0.964	0.757	0.748	3.3 (1.8)
FT-T	**0.459**	**0.859**	0.391	**0.732**	**0.729**	0.960	**0.8982**	8.855	**0.970**	0.756	0.746	1.8 (1.2)

Gorishniy et al. (2021)

Results

Comparison with GBDT takeaways:

- FT-Transformer allows building powerful ensembles out of the box.
- There is still no universal solution among DL models and GBDT.
- DL research efforts aimed at surpassing GBDT should focus on datasets where GBDT outperforms state-of-the-art DL solutions.
- Including “DL-friendly” problems is still important to avoid degradation on such problems.

	CA ↓	AD ↑	HE ↑	JA ↑	HI ↑	AL ↑	EP ↑	YE ↓	CO ↑	YA ↓	MI ↓
Default hyperparameters											
XGBoost	0.462	**0.874**	0.348	0.711	0.717	0.924	0.8799	9.192	0.964	0.761	0.751
CatBoost	**0.428**	0.873	0.386	0.724	0.728	0.948	0.8893	8.885	0.910	0.749	0.744
FT-Transformer	0.454	0.860	**0.395**	**0.734**	**0.731**	**0.966**	**0.8969**	**8.727**	**0.973**	**0.747**	**0.742**
Tuned hyperparameters											
XGBoost	0.431	0.872	0.377	0.724	0.728	–	0.8861	8.819	0.969	**0.732**	0.742
CatBoost	**0.423**	**0.874**	0.388	0.727	0.729	–	0.8898	8.837	0.968	0.740	**0.741**
ResNet	0.478	0.857	0.398	0.734	0.731	0.966	0.8976	8.770	0.967	0.751	0.745
FT-Transformer	0.448	0.860	**0.398**	**0.739**	**0.731**	**0.967**	**0.8984**	**8.751**	**0.973**	0.747	0.743

Gorishniy et al. (2021)

Explainability

In-build explainability of FT-Transformer

$$p = \frac{1}{n_{samples}} \sum_i p_i \qquad p_i = \frac{1}{n_{heads} \times L} \sum_{h,l} p_{ihl}.$$

Feature importances formula

p_{ihl} is the h-th head's attention map for the [CLS] token from the forward pass of the l-th layer on the i-th sample

Interpretation:

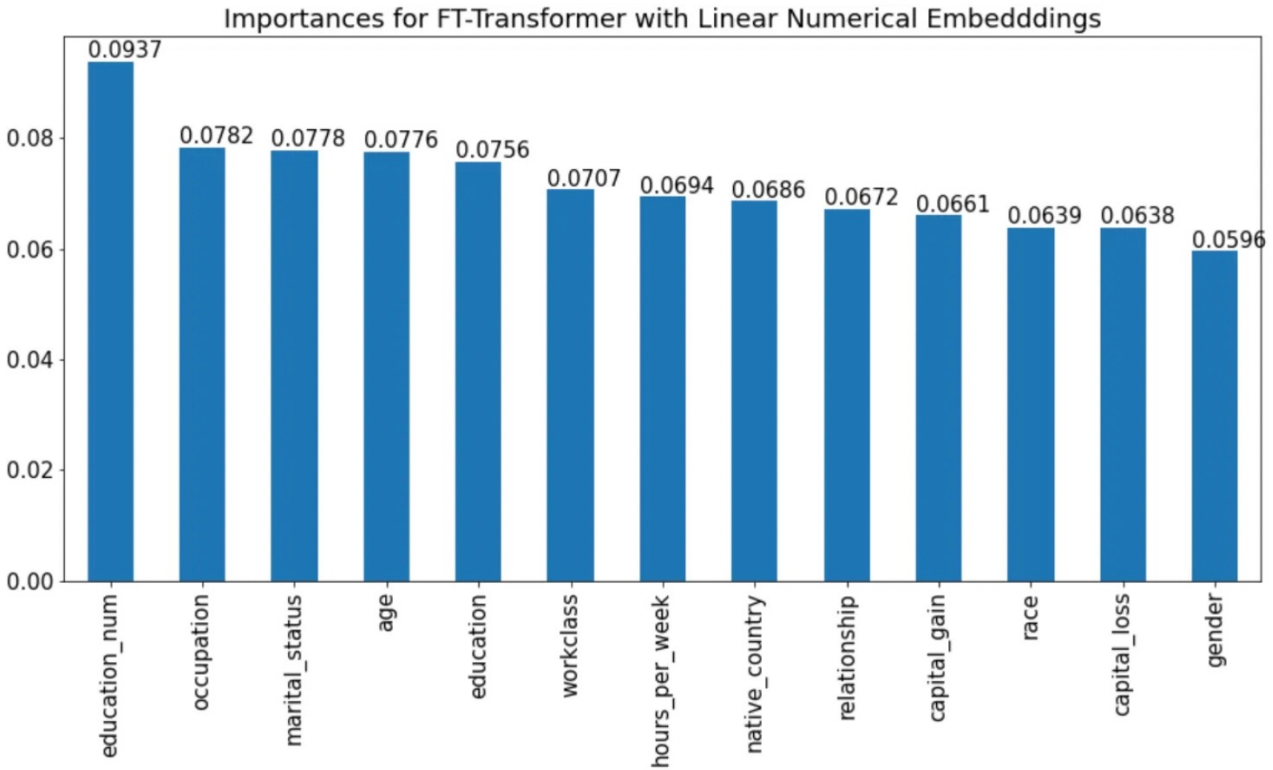
- sum up all the attention scores for [CLS] token across different attention-heads and Transformer layers
- divide them by *heads x depth*
- average local importances (p_i) across all rows to get the global importances (p)

Gorishniy et al. (2021)

Experiments

[Adult Income Dataset]

Task: Predict whether income exceeds $50K based on census data



Feature importances.

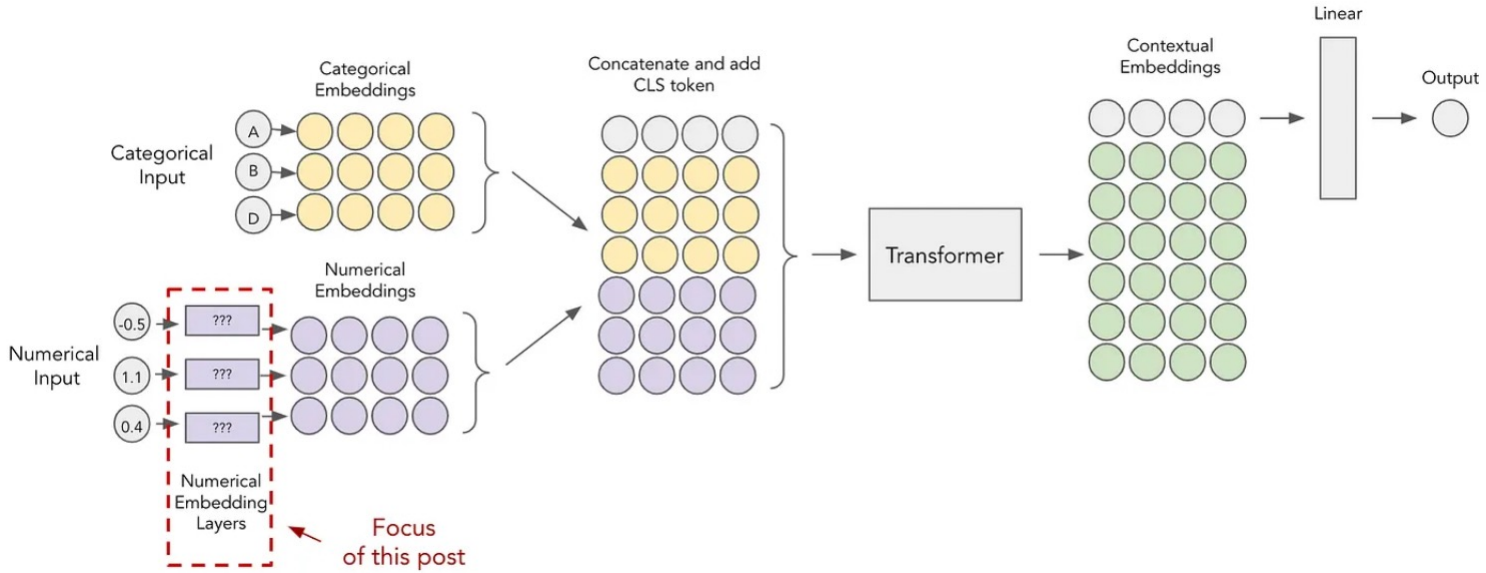
Piecewise Linear & Periodic Encodings

Based on the series of posts https://towardsdatascience.com/transformers-for-tabular-data-part-3-piecewise-linear-periodic-encodings-1fc49c4bd7bc

Motivation

Previously we only considered linear embeddings of numerical features.

Question: is it optimal?



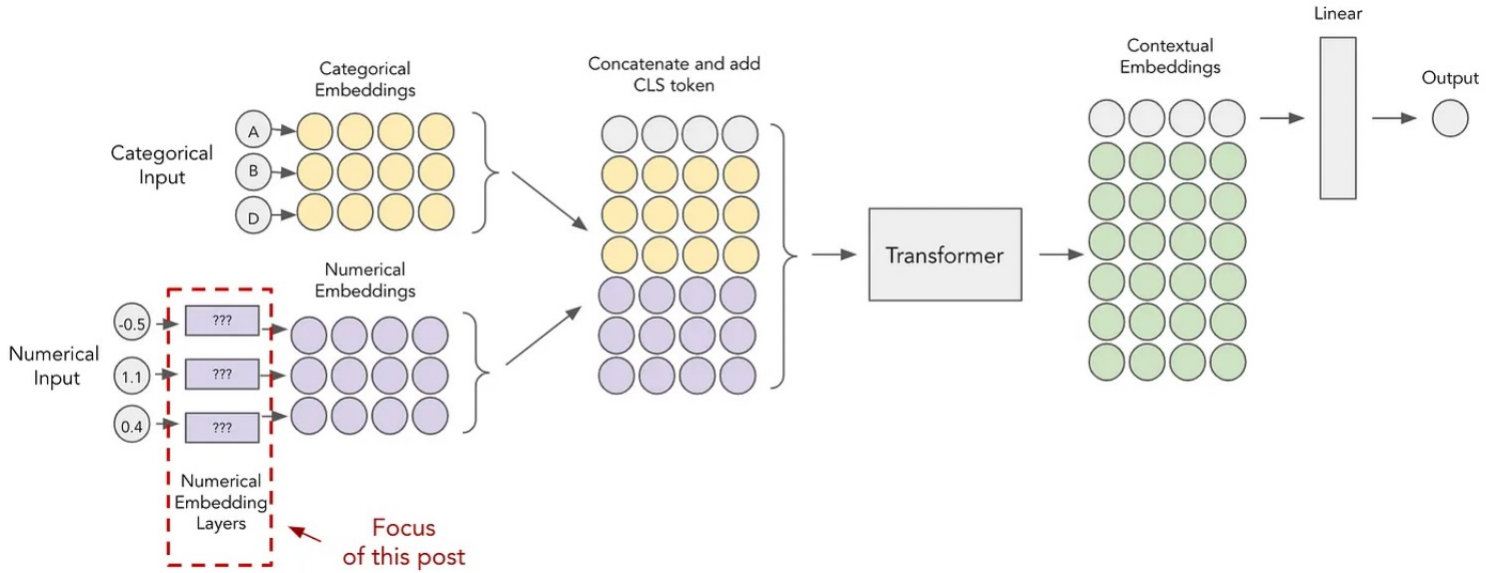
MLP with categorical embeddings.

Motivation

Previously we only considered linear embeddings of numerical features.

Question: is it optimal?

Idea: Piecewise Linear Encoding and Periodic Encoding



Gorishniy et al. (2022) in [On Embeddings for Numerical Features in Tabular Deep Learning]

On Embeddings for Numerical Features in Tabular Deep Learning

Yury Gorishniy Ivan Rubachev Artem Babenko

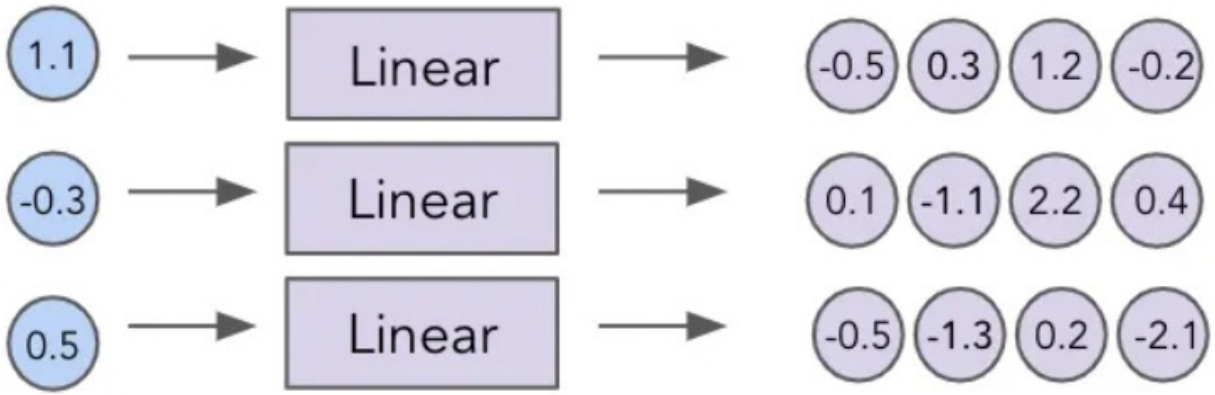
Abstract

gap between them and the "shallow" ensembles of decision trees, like GBDT, often remains significant (Gorishniy et al.,

Linear Embeddings

Previously:
Linear embedding layers are simple fully connected layers with optional ReLU activation.

These layers do not share weights between each others, so there's one embedding layer per numerical feature.



Linear Numerical Embeddings.

Periodic Embeddings

Pipeline:

1. Transformation into pre-activation values (***v***) using a learned vector (***c***)

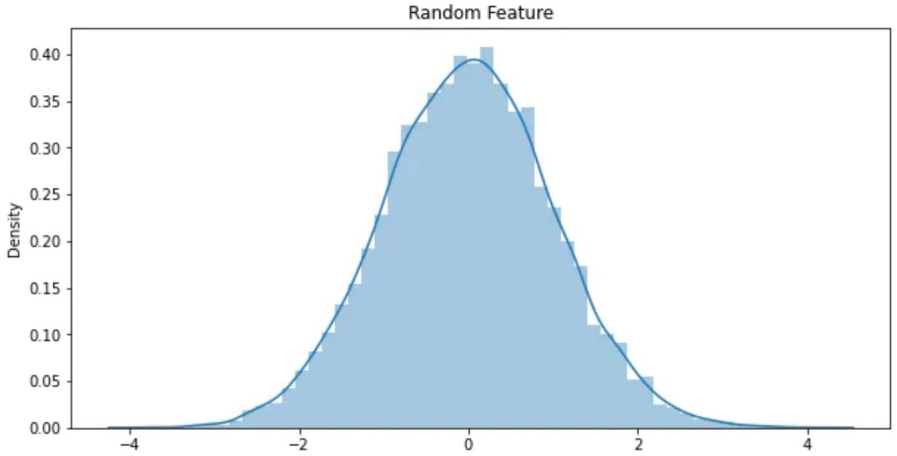
2. Activation of values (***v***) using Sine and Cosine

3. Concatenation of Sine and Cosine values

$$f_i(x) = \texttt{Periodic}(x) = \texttt{concat}[\sin(v),\ \cos(v)],$$
$$v = [2\pi c_1 x,\ \ldots,\ 2\pi c_k x]$$

Periodic Embeddings

Example:

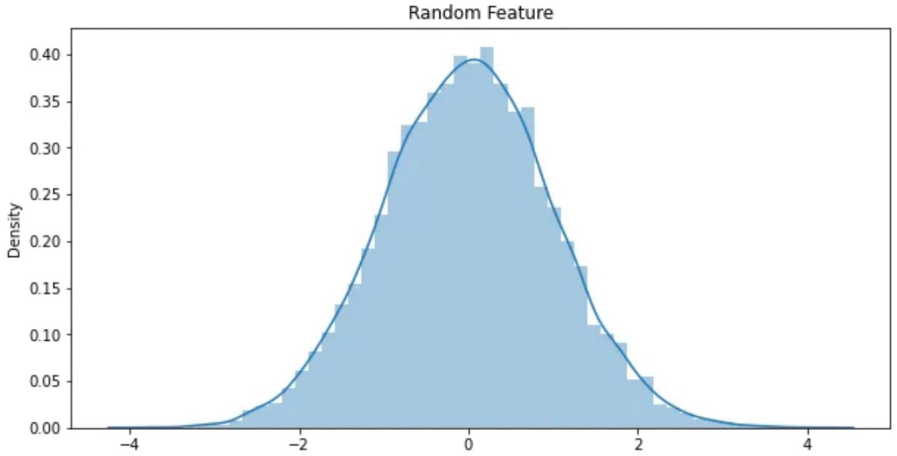


Random feature distribution

$$f_i(x) = \texttt{Periodic}(x) = \texttt{concat}[\sin(v),\ \cos(v)],$$
$$v = [2\pi c_1 x,\ \ldots,\ 2\pi c_k x]$$

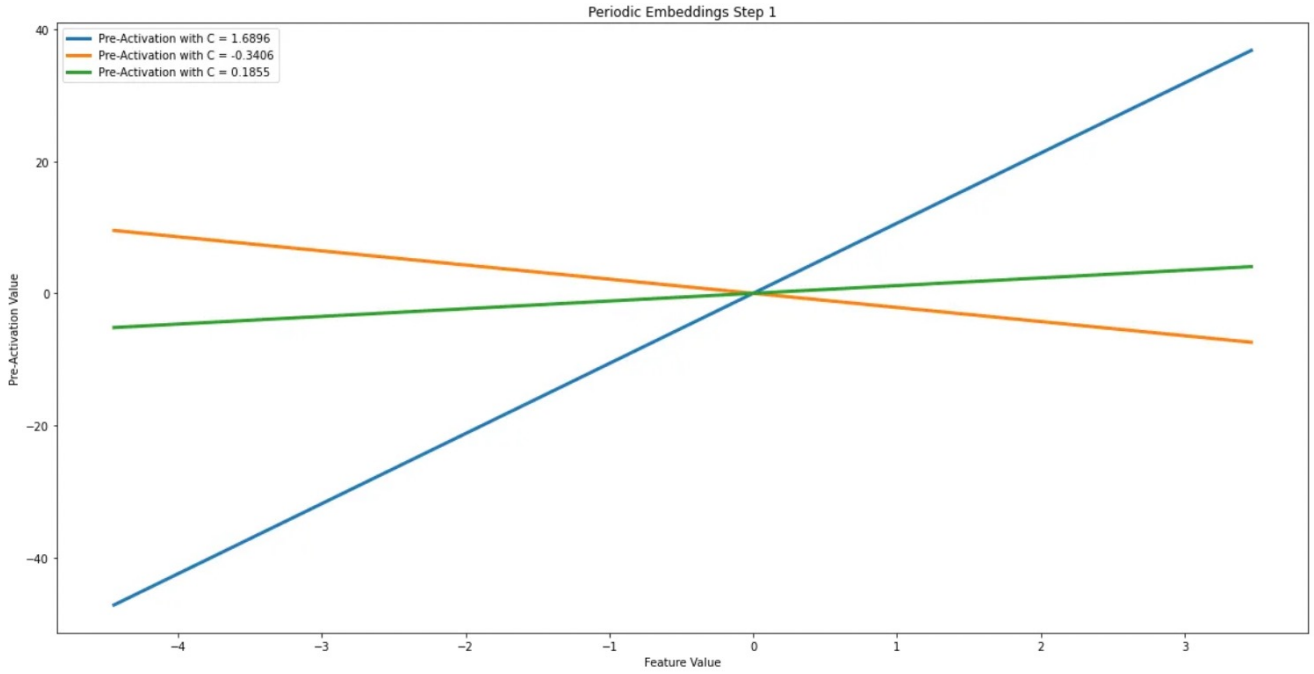
Periodic Embeddings

Example:



Random feature distribution

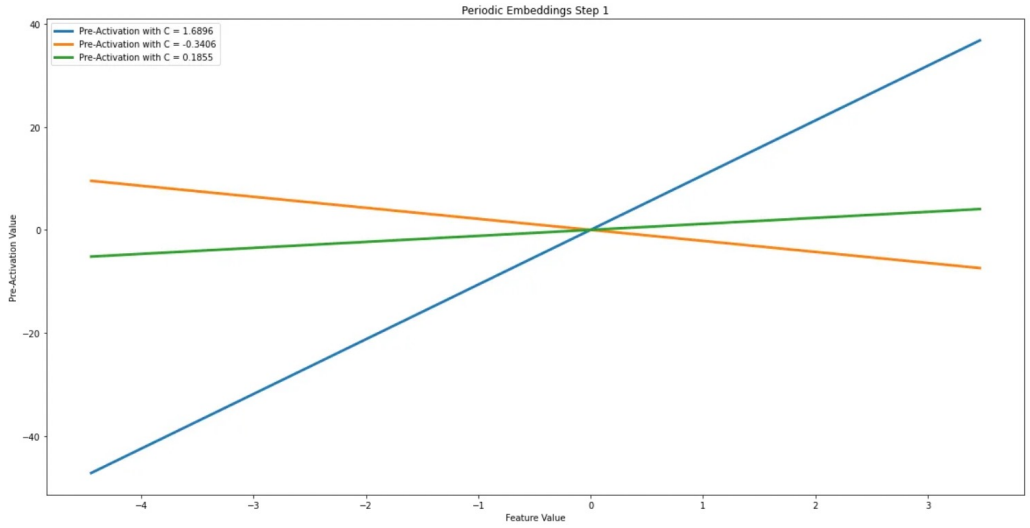
$$f_i(x) = \texttt{Periodic}(x) = \texttt{concat}[\sin(v),\ \cos(v)],$$
$$v = [2\pi c_1 x,\ \ldots,\ 2\pi c_k x]$$



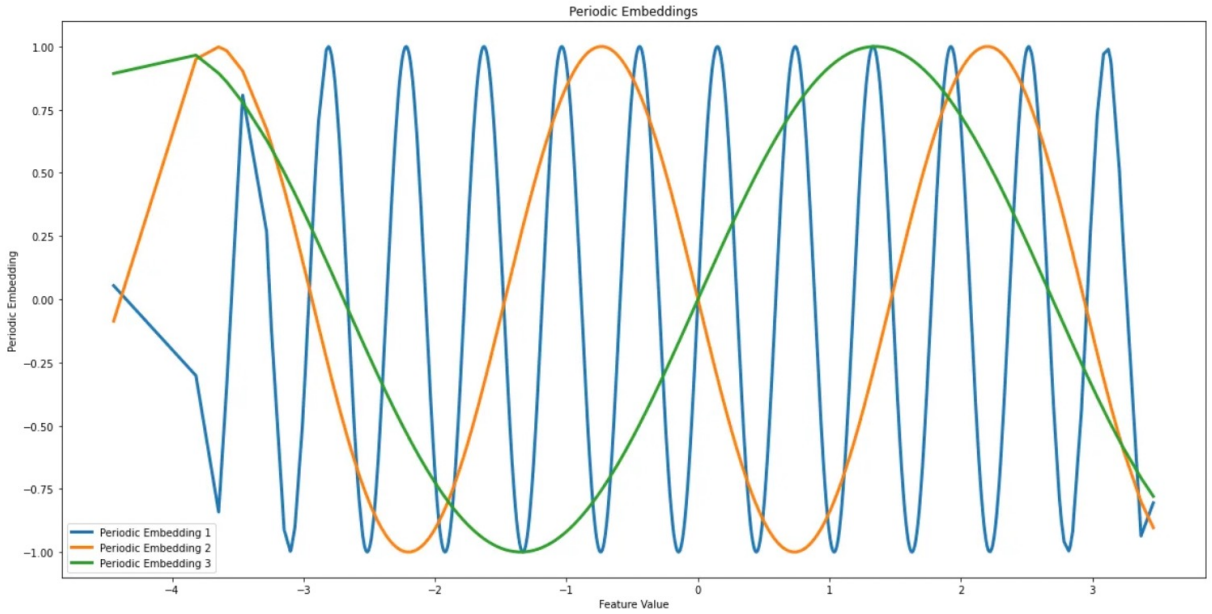
Periodic pre-activation embedding of the random feature for different C

Periodic Embeddings

Example:



Periodic pre-activation embedding of the random feature for different C



Periodic post-activation embeddings of the random feature

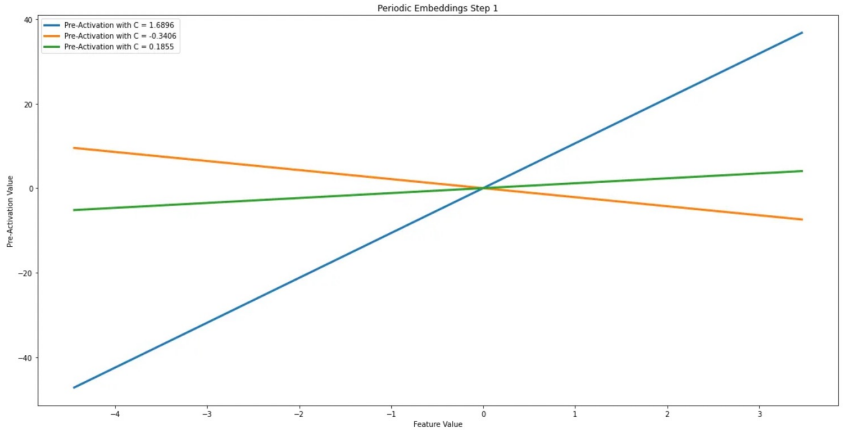
$$f_i(x) = \texttt{Periodic}(x) = \texttt{concat}[\sin(v),\ \cos(v)],$$
$$v = [2\pi c_1 x,\ \ldots,\ 2\pi c_k x]$$

Periodic Embeddings

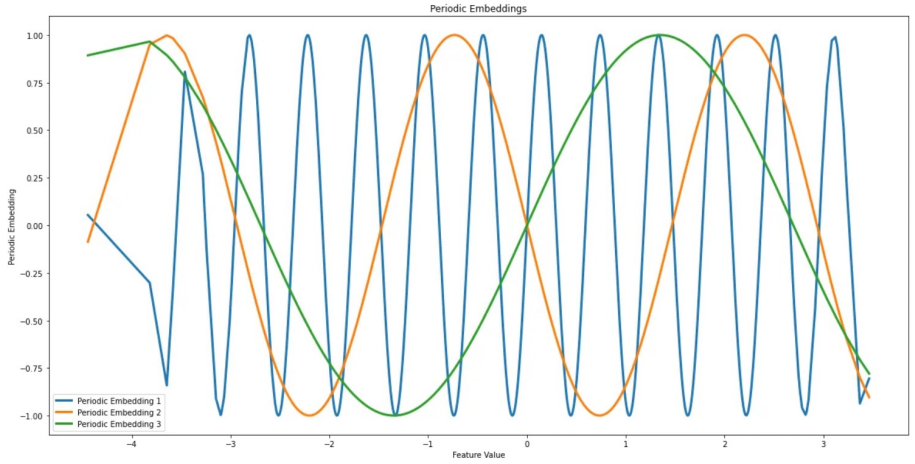
Example:

Takeaways:

- the slope affects periodic activations frequency
- larger slope (blue line) have post-activation values that have higher frequency
- small slope (green and orange lines) result in low-frequency activations.



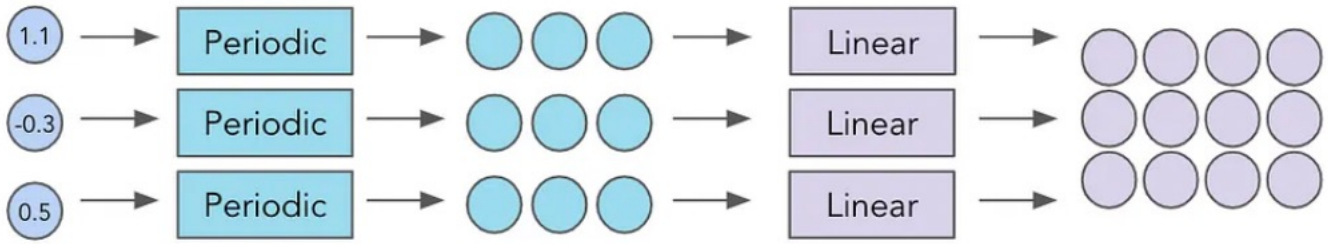
Periodic pre-activation embedding of the random feature for different C



Periodic post-activation embeddings of the random feature

Method

Add an additional linear layer on top of the periodic encoding



Periodic embedding

Piecewise Linear Encoding (PLE)

Adapts the idea of one-hot-encoding.

First we split a feature into ***t*** bins.

Two splitting methods:

- *quantile binning*
- *target binning*

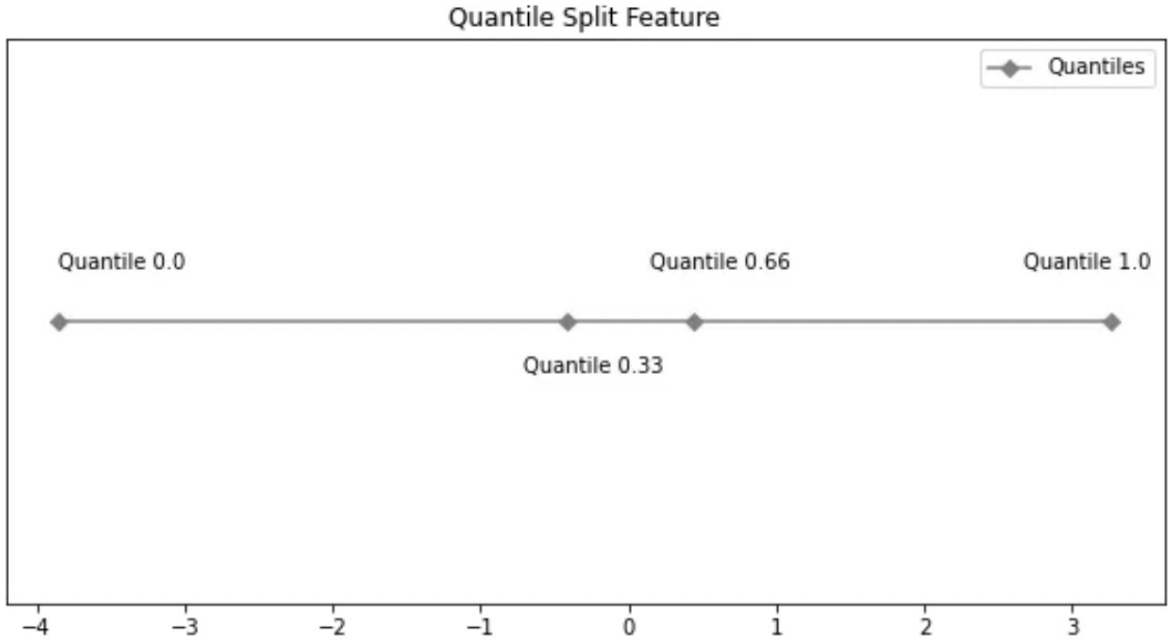
Quantile Binning

Idea: we split our feature into ***t*** equal quantile bins.

Quantile Binning Approach

Idea: we split our feature into ***t*** equal width bins.

Example: three bins (i.e. ***t*** = *3*) => quantiles are *0, 0.33, 0.66, 1.0.*



Quantile binning of the random feature.

Quantile Binning Approach

Idea: we split our feature into t equal width bins.

Each quantile is represented as a tuple *[bin_start, bin_end)*

$$B_t^i = [b_{t-1}^i, b_t^i)$$

Bins formula notation

Piecewise Linear Encoding formula:

$$\mathrm{PLE}(x) = [e_1, \ldots, e_T] \in \mathbb{R}^T$$

$$e_t = \begin{cases} 0, & x < b_{t-1} \text{ AND } t > 1 \\ 1, & x \geq b_t \text{ AND } t < T \\ \frac{x - b_{t-1}}{b_t - b_{t-1}}, & \text{otherwise} \end{cases}$$

Gorishniy et al. (2022)

Quantile Binning Approach

Idea: we split our feature into t equal width bins.

Each quantile is represented as a tuple *[bin_start, bin_end)*

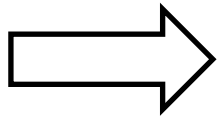
$$B_t^i = [b_{t-1}^i, b_t^i)$$

Bins formula notation

Piecewise Linear Encoding formula:

$$\mathrm{PLE}(x) = [e_1, \ldots, e_T] \in \mathbb{R}^T$$

$$e_t = \begin{cases} 0, & x < b_{t-1} \text{ AND } t > 1 \\ 1, & x \geq b_t \text{ AND } t < T \\ \frac{x - b_{t-1}}{b_t - b_{t-1}}, & \text{otherwise} \end{cases}$$



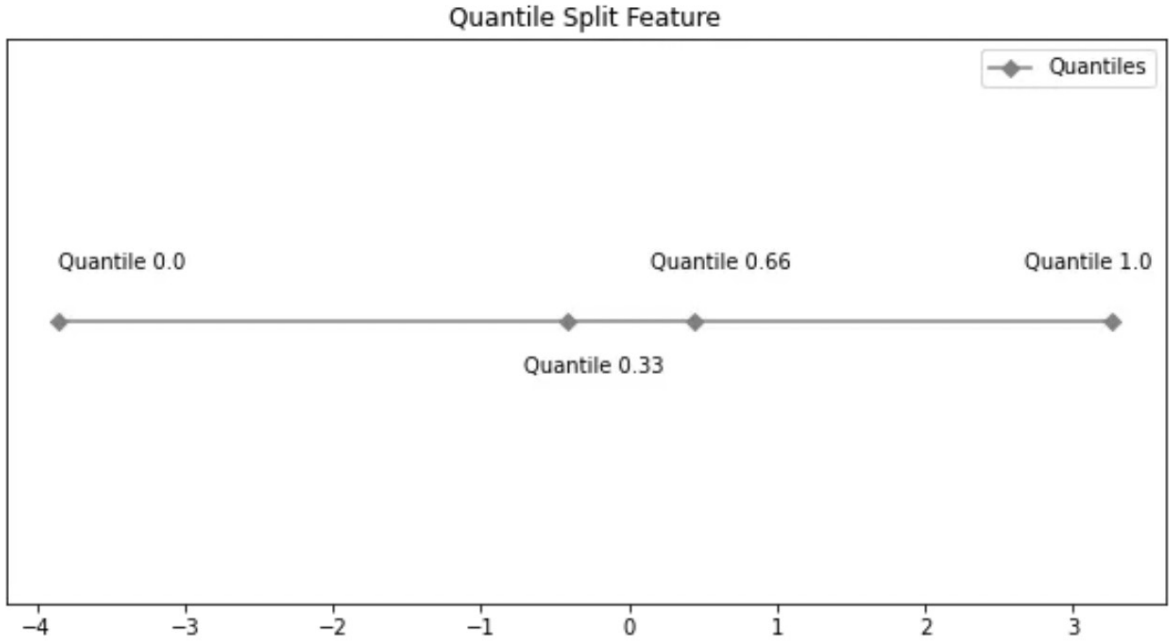
t dimensional embedding for each value

Gorishniy et al. (2022)

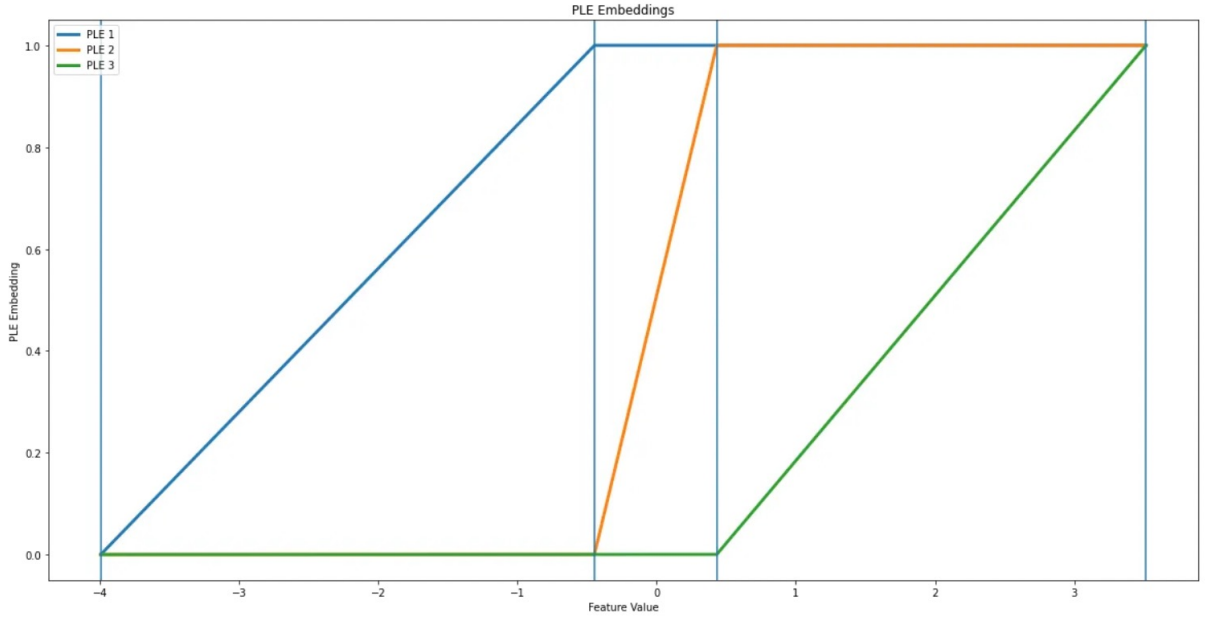
Quantile Binning Approach

Idea: we split our feature into t equal width bins.

Example: three bins (i.e. ***t*** = *3*)

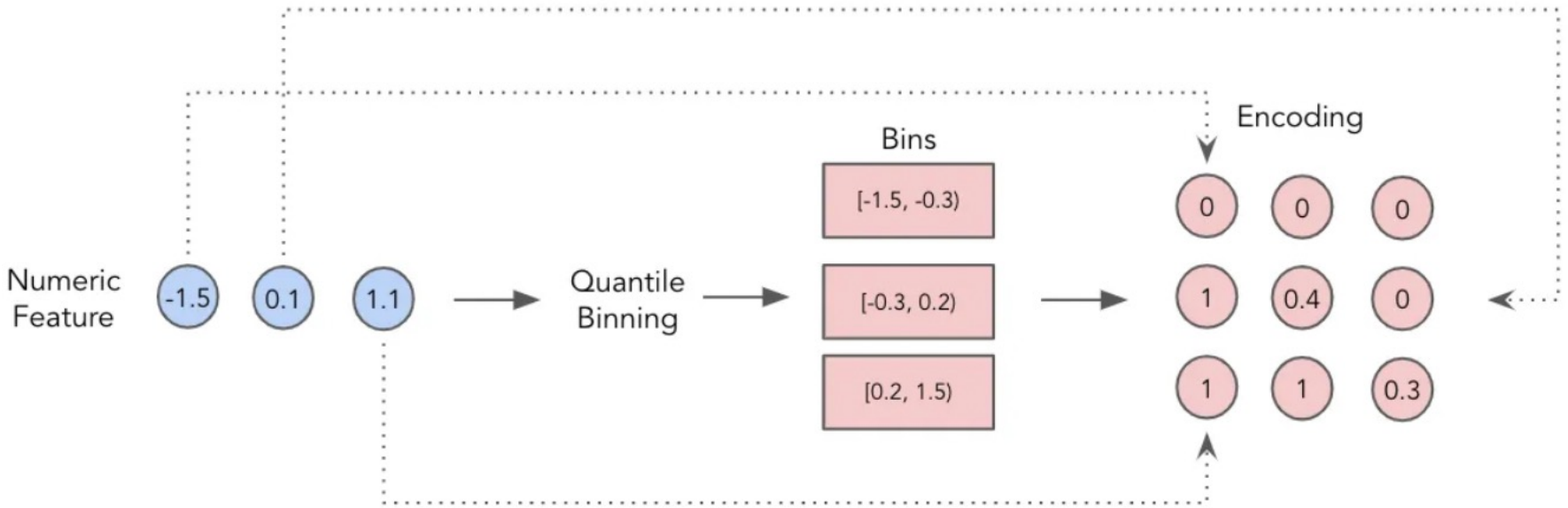


Quantile binning of the random feature.



Its PLE embeddings.

Method



PLE embeddings with quantile binning.

Target Binning Approach

Idea: use a Decision Tree (DT) to construction the bins.

Why?

Target Binning Approach

Idea: use a Decision Tree (DT) to construction the bins.

Why?

Motivation: quantile approach splits our feature into the bins of equal width but it might be suboptimal in certain cases.

Target Binning Approach

Idea: use a Decision Tree (DT) to construction the bins.

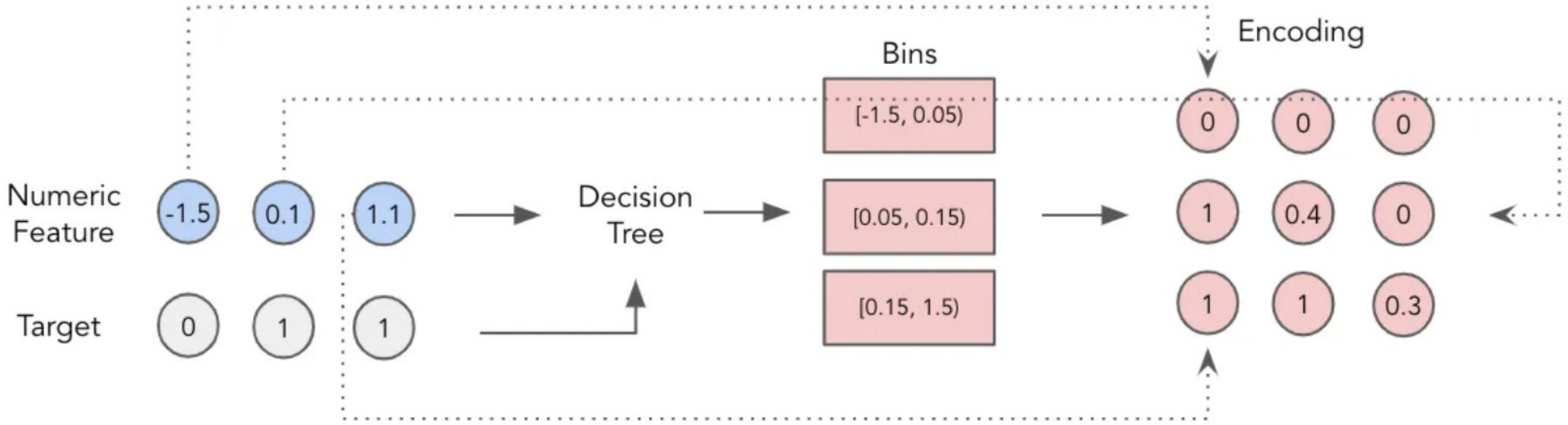
Why?

Motivation: quantile approach splits our feature into the bins of equal width but it might be suboptimal in certain cases.

Solution: DT finds the most meaningful splits with regards to the target.

Example: *if there is more target variance towards the larger values of the feature, majority of the bins move to the right.*

Method



PLE embeddings with target binning.

Experiments

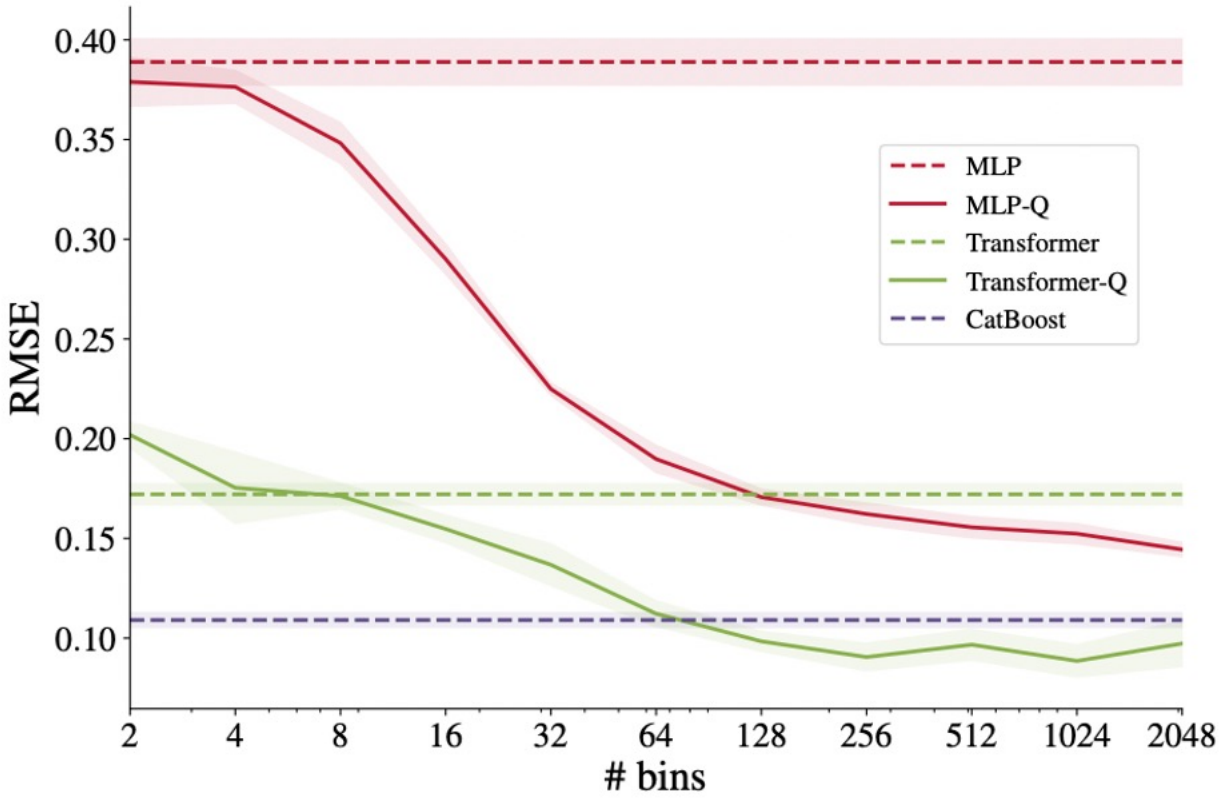


Figure 2. RMSE (averaged over five random seeds) of different approaches on the same synthetic GBDT-friendly task. Using `PLE`-representations ("-Q") instead of scalar values improves the performance of MLP and Transformer. Note that in practice, increasing the number of bins does not always lead to better results.

Gorishniy et al. (2022)

Experiments

Takeaways:

- there's no single winner across the dataset
- embedding is another hyperparameter to tune
- most of the times Periodic & PLE encodings help to improve the score

Name	Embedding function (f_i)
L	Linear
LR	ReLU ∘ Linear
LRLR	ReLU ∘ Linear ∘ ReLU ∘ Linear
Q	PLE_q
Q-L	Linear ∘ PLE_q
Q-LR	ReLU ∘ Linear ∘ PLE_q
Q-LRLR	ReLU ∘ Linear ∘ ReLU ∘ Linear ∘ PLE_q
T	PLE_t
T-L	Linear ∘ PLE_t
T-LR	ReLU ∘ Linear ∘ PLE_t
T-LRLR	ReLU ∘ Linear ∘ ReLU ∘ Linear ∘ PLE_t
P	Periodic
PL	Linear ∘ Periodic
PLR	ReLU ∘ Linear ∘ Periodic
PLRLR	ReLU ∘ Linear ∘ ReLU ∘ Linear ∘ Periodic
AutoDis	Linear ∘ SoftMax ∘ Linear_ ∘ LReLU ∘ Linear_

	GE ↑	CH ↑	EY ↑	CA ↓	HO ↓	AD ↑	OT ↑	HI ↑	FB ↓	SA ↑	CO ↑	MI ↓
MLP	0.632	0.856	0.615	0.495	3.204	0.854	0.818	0.720	5.686	0.912	0.964	**0.747**
MLP-Q	**0.653**	0.854	0.604	0.464	**3.163**	0.859	0.816	0.721	5.766	0.922	0.968	0.750
MLP-T	**0.647**	**0.861**	0.682	0.447	**3.149**	0.864	**0.821**	0.720	5.577	0.923	0.967	0.749
MLP-Q-LR	**0.646**	0.857	**0.693**	0.455	**3.184**	0.863	0.811	0.720	**5.394**	0.923	**0.969**	**0.747**
MLP-T-LR	0.640	**0.861**	**0.685**	**0.439**	3.207	**0.868**	0.818	**0.724**	**5.508**	**0.924**	**0.968**	0.747
Transformer-L	0.632	0.860	0.731	0.465	3.239	0.858	**0.817**	0.725	**5.602**	0.924	0.971	**0.746**
Transformer-Q-L	**0.659**	0.856	0.753	0.451	3.319	0.867	0.812	**0.729**	5.741	**0.924**	**0.973**	0.747
Transformer-T-L	**0.663**	**0.861**	**0.775**	0.454	**3.197**	**0.871**	**0.817**	0.726	5.803	**0.924**	**0.974**	0.747
Transformer-Q-LR	**0.659**	0.857	**0.796**	0.448	3.270	0.867	0.812	0.723	5.683	0.923	0.972	0.748
Transformer-T-LR	**0.665**	0.860	**0.789**	**0.442**	**3.219**	0.870	**0.818**	**0.729**	5.699	**0.924**	0.973	0.747

Gorishniy et al. (2022)

Experiments

Comparison with other models

	GE ↑	CH ↑	EY ↑	CA ↓	HO ↓	AD ↑	OT ↑	HI ↑	FB ↓	SA ↑	CO ↑	MI ↓	Avg. Rank
CatBoost	0.692	0.861	0.757	0.430	3.093	0.873	0.825	0.727	5.226	0.924	0.967	**0.741**	6.8 ± 4.9
XGBoost	0.683	0.859	0.738	0.434	3.152	**0.875**	0.827	0.726	5.338	0.919	0.969	0.742	9.0 ± 5.7
MLP	0.665	0.856	0.637	0.486	3.109	0.856	0.822	0.727	5.616	0.913	0.968	0.746	15.6 ± 2.4
MLP-LR	0.679	0.861	0.694	0.463	3.012	0.859	0.826	0.731	5.477	0.924	0.972	0.744	10.2 ± 4.4
MLP-Q-LR	0.682	0.859	0.732	0.433	3.080	0.867	0.818	0.724	**5.144**	0.924	0.974	0.745	10.7 ± 4.6
MLP-T-LR	0.673	0.861	0.729	0.435	3.099	0.870	0.821	0.727	5.409	0.924	0.973	0.746	10.3 ± 3.8
MLP-PLR	**0.700**	0.858	0.968	0.453	**2.975**	0.874	**0.830**	**0.734**	5.388	**0.924**	0.975	0.743	4.9 ± 4.8
ResNet	0.690	0.861	0.667	0.483	3.081	0.856	0.821	0.734	5.482	0.918	0.968	0.745	12.1 ± 4.7
ResNet-LR	0.672	0.862	0.735	0.450	2.992	0.859	0.822	0.733	5.415	0.923	0.971	0.743	9.8 ± 4.3
ResNet-Q-LR	0.674	0.859	0.794	0.427	3.066	0.868	0.815	0.729	5.309	0.923	0.976	0.746	9.2 ± 4.8
ResNet-T-LR	0.683	0.862	0.817	**0.425**	3.030	0.872	0.822	0.731	5.471	0.923	0.975	0.744	7.8 ± 3.6
ResNet-PLR	0.691	0.861	0.925	0.443	3.040	**0.874**	0.825	0.734	5.400	0.924	0.975	0.743	5.2 ± 2.3
Transformer-L	0.668	0.861	0.769	0.455	3.188	0.860	0.824	0.727	5.434	0.924	0.973	0.743	10.6 ± 3.3
Transformer-LR	0.666	0.861	0.776	0.446	3.193	0.861	0.824	0.733	5.430	0.924	0.973	0.743	9.4 ± 4.1
Transformer-Q-LR	0.690	0.857	0.842	**0.425**	3.143	0.868	0.818	0.726	5.471	**0.924**	0.975	0.744	8.5 ± 5.5
Transformer-T-LR	0.686	0.862	0.833	**0.423**	3.149	0.871	0.823	0.733	5.515	0.924	**0.976**	0.744	7.2 ± 4.6
Transformer-PLR	0.686	**0.864**	**0.977**	0.449	3.091	0.873	0.823	0.734	5.581	**0.924**	0.975	0.743	6.0 ± 4.5

Gorishniy et al. (2022)

Conclusion

Summary

- Transformers can be use to contextualize categorical embeddings
- To contextualize numerical features with Transformers they should be first embedded
- Linear embeddings is the simplest method to obtain numerical embeddings
- We may use more advanced numerical embedding techniques such as Periodic Encoding & Linear Piecewise Encoding

Questions?

Bonus: TabNet

Based on the post https://towardsdatascience.com/tabnet-e1b979907694

Motivation

- TabNet is a DL architecture which **outperformed the leading tree based models** across a variety of benchmarks.
- Is more explainable than boosted tree models as it has **built-in explainability**.
- It can also be used **without any feature preprocessing**.

Sercan O. Arık, Tomas Pfister in TabNet: Attentive Interpretable [TabNet: Attentive Interpretable Tabular Learning]

TabNet: Attentive Interpretable Tabular Learning

Sercan Ö. Arık, Tomas Pfister
Google Cloud AI
Sunnyvale, CA
soarik@google.com, tpfister@google.com

What is TabNet?

1. Inputs raw tabular data **without any preprocessing**

What is TabNet?

1. Inputs raw tabular data **without any preprocessing**

2. Is trained using **gradient descent**-based optimization.

What is TabNet?

1. Inputs raw tabular data **without any preprocessing**

2. Is trained using **gradient descent**-based optimization.

3. Uses **sequential attention** to choose features at each decision step, enabling **interpretability** and better learning as the **learning capacity is used for the most useful features.**

What is TabNet?

1. Inputs raw tabular data **without any preprocessing**

2. Is trained using **gradient descent**-based optimization.

3. Uses **sequential attention** to choose features at each decision step, enabling **interpretability** and better learning as the **learning capacity is used for the most useful features.**

4. Performs **instance-wise feature selection is**, e.g. it can be different for each row of the training dataset.

What is TabNet?

1. Inputs raw tabular data **without any preprocessing**

2. Is trained using **gradient descent**-based optimization.

3. Uses **sequential attention** to choose features at each decision step, enabling **interpretability** and better learning as the **learning capacity is used for the most useful features.**

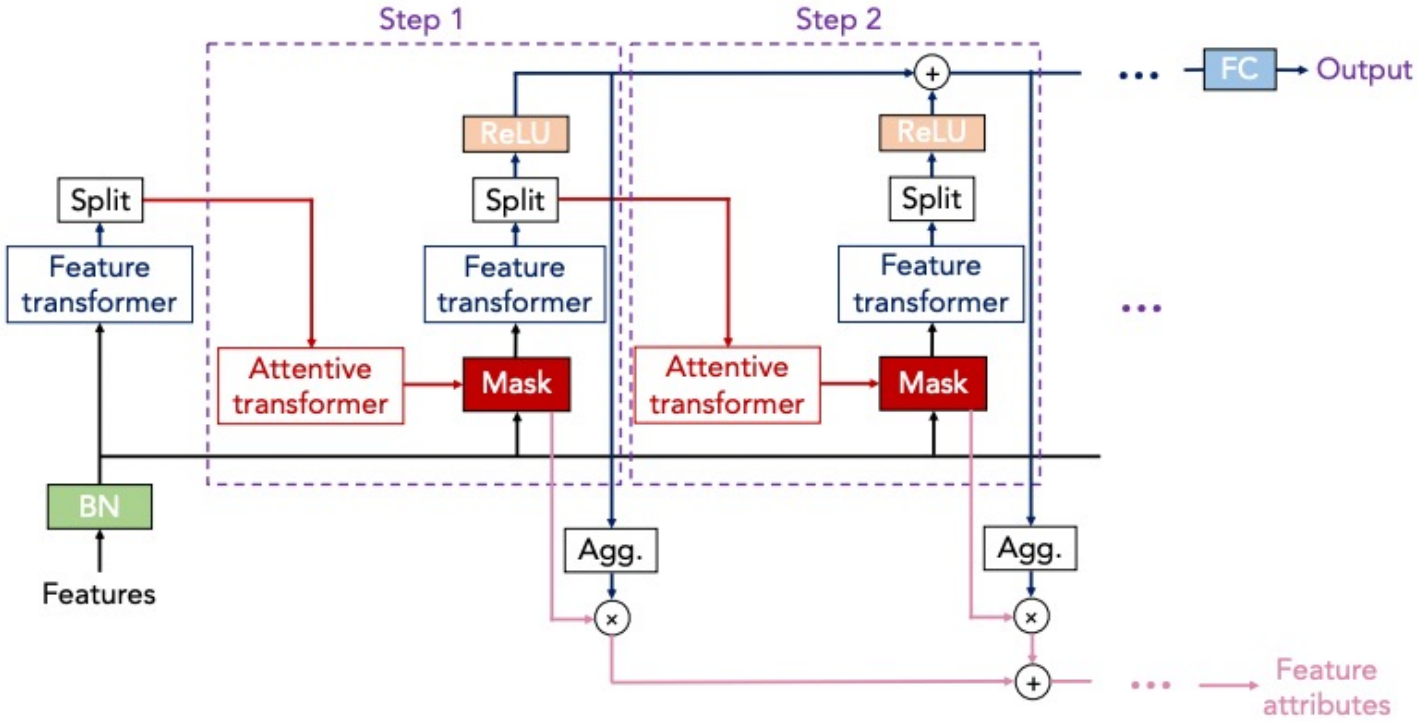
4. Performs **instance-wise feature selection is**, e.g. it can be different for each row of the training dataset.

5.Enables two kinds of interpretability:

- ***local interpretability*** *that visualizes the importance of features and how they are combined for a single row*
- ***global interpretability*** *which quantifies the contribution of each feature to the trained model across the dataset.*

TabNet high-level

- Each **Step** is a block of components.
- The number of **Steps** is a hyperparameter.
- Each **Step** gets its own vote in the final classification and these votes are equally weighted.
- This mimics an *ensemble classification.*

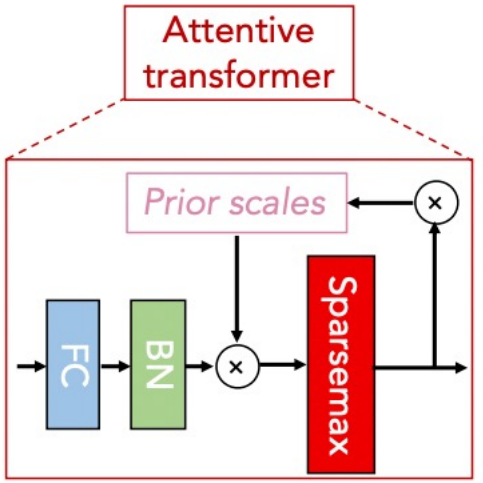


TabNet encoder architecture

Sercan O. Arık, Tomas Pfister in TabNet: Attentive Interpretable TabNet: Attentive Interpretable Tabular Learning

Attentive transformer

- Transformed features are passed to the **Attentive Transformer** and the **Mask** for feature selection.
- The **Attentive Transformer** is comprised of a *FC layer, BN* and *Sparsemax* normalisation.
- It also includes **prior scales**, meaning it **knows how much each feature has been used by the previous steps**.

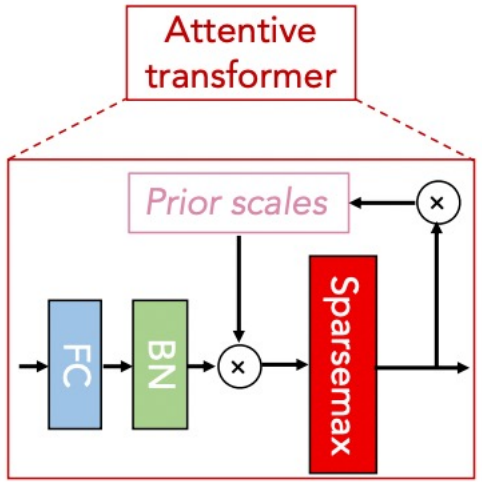


Attentive transformer

Sercan O. Arık, Tomas Pfister in TabNet: Attentive Interpretable TabNet: Attentive Interpretable Tabular Learning

Attentive transformer & Mask

- Transformed features are passed to the **Attentive Transformer** and the **Mask** for feature selection.
- The **Attentive Transformer** is comprised of a *FC layer, BN* and *Sparsemax* normalisation.
- It also includes **prior scales**, meaning it **knows how much each feature has been used by the previous steps**.
- This is used to derive the **Mask** using the processed features from the previous **Feature Transformer**.



Attentive transformer

Sercan O. Arık, Tomas Pfister in TabNet: Attentive Interpretable TabNet: Attentive Interpretable Tabular Learning

Mask

Details:

- TabNet employs **soft feature selection with controllable sparsity** in end-to-end learning => one model jointly performs feature selection and output mapping, which leads to better performance.

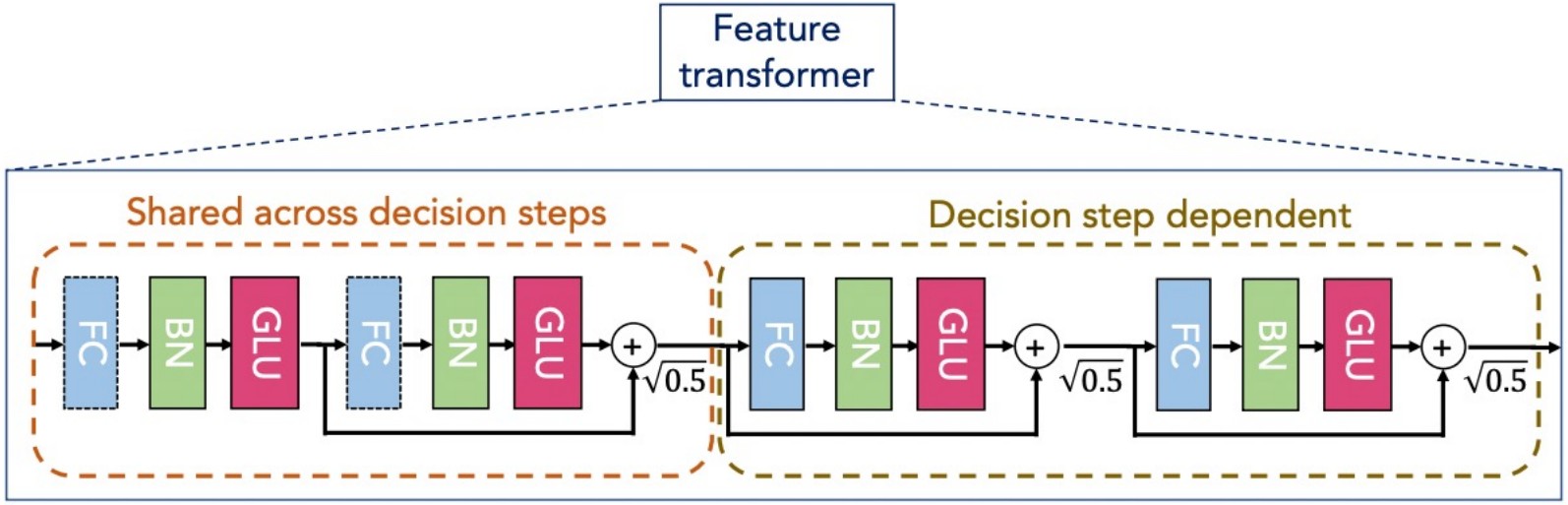
- TabNet uses **instance-wise feature selection**, which means features are selected for each input and each prediction can use different features.

Sercan O. Arık, Tomas Pfister in TabNet: Attentive Interpretable TabNet: Attentive Interpretable Tabular Learning

Feature Transformer

The **Feature Transformer** has multiple layers, some of which are shared across every **Step** while others are unique to each **Step**.

Each layer is composed of a fully-connected (FC) layer, BN and GLU nonlinearity.



Feature Transformer

Sercan O. Arık, Tomas Pfister in TabNet: Attentive Interpretable TabNet: Attentive Interpretable Tabular Learning

Results

Table 2: Performance for Forest Cover Type dataset.

Model	*Test accuracy (%)*
XGBoost	89.34
LightGBM	89.28
CatBoost	85.14
AutoML Tables	94.95
TabNet	**96.99**

Table 4: Performance on Sarcos dataset. Three TabNet models of different sizes are considered.

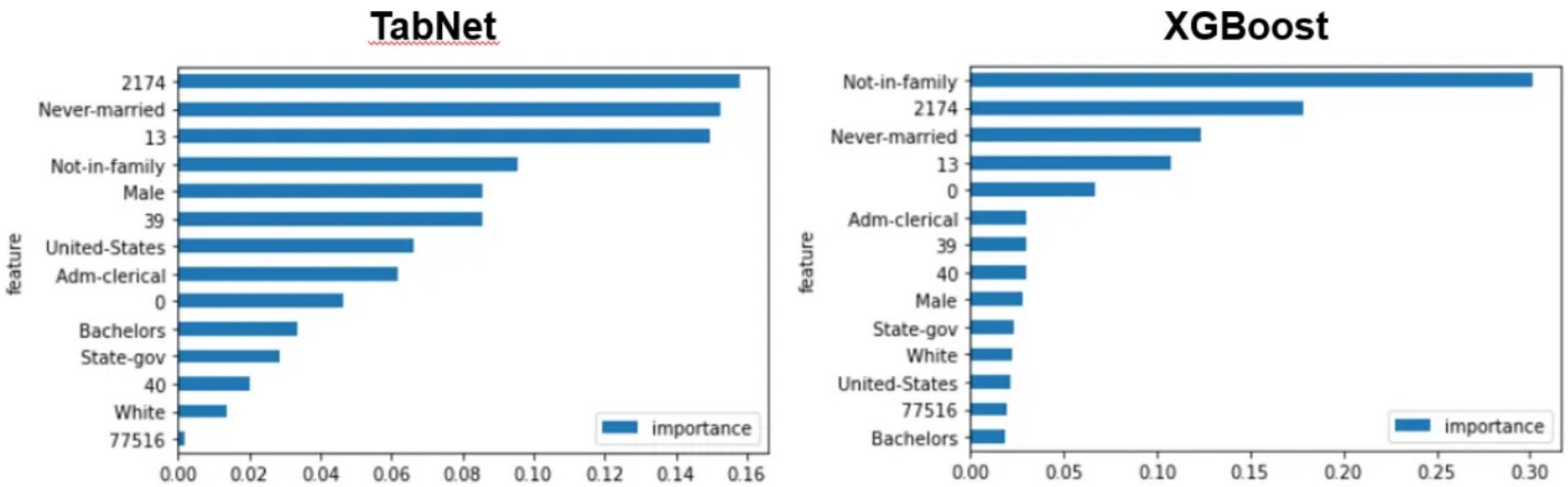
Model	*Test MSE*	*Model size*
Random forest	2.39	16.7K
Stochastic DT	2.11	28K
MLP	2.13	0.14M
Adaptive neural tree	1.23	0.60M
Gradient boosted tree	1.44	0.99M
TabNet-S	**1.25**	**6.3K**
TabNet-M	**0.28**	**0.59M**
TabNet-L	**0.14**	**1.75M**

Table 5: Performance on Higgs Boson dataset. Two TabNet models are denoted with -S and -M.

Model	*Test acc. (%)*	*Model size*
Sparse evolutionary MLP	**78.47**	**81K**
Gradient boosted tree-S	74.22	0.12M
Gradient boosted tree-M	75.97	0.69M
MLP	78.44	2.04M
Gradient boosted tree-L	76.98	6.96M
TabNet-S	78.25	81K
TabNet-M	**78.84**	**0.66M**

Sercan O. Arık, Tomas Pfister in TabNet: Attentive Interpretable TabNet: Attentive Interpretable Tabular Learning

Feature importances



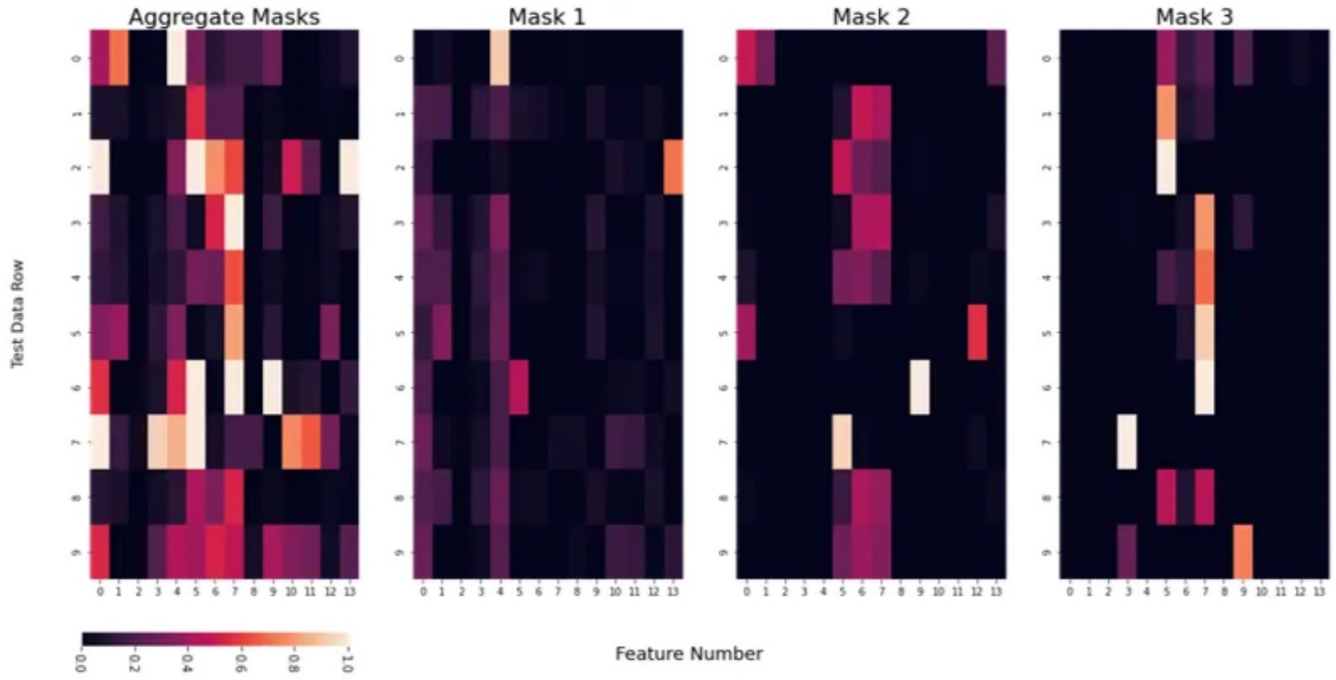
Feature Importances from TabNet and XGBoost models trained on a census dataset.

Feature importances

By using the masks, we can understand which features are being used at a prediction level, we can look at the aggregate of all the **Masks** or an individual **Mask**.

⇩

Some understanding of which features the model has used to make its prediction.



Mask visualization from TabNet model trained on a census dataset.

Bonus: linear embeddings help to improve the score

	GE ↑	CH ↑	EY ↑	CA ↓	HO ↓	AD ↑	OT ↑	HI ↑	FB ↓	SA ↑	CO ↑	MI ↓
MLP	0.632	0.856	0.615	0.495	3.204	0.854	0.818	0.720	5.686	0.912	**0.964**	0.747
MLP-L	**0.639**	**0.861**	0.635	0.475	3.123	0.856	**0.820**	0.723	5.684	0.916	0.963	0.748
MLP-LR	**0.642**	**0.860**	**0.660**	**0.471**	**3.084**	0.857	**0.819**	**0.726**	**5.625**	0.923	0.963	**0.746**
MLP-AutoDis	**0.649**	0.857	0.634	0.474	3.165	**0.859**	0.807	**0.725**	**5.670**	**0.924**	**0.963**	–

L – Linear
LR – ReLU ◦ Linear
AutoDis – Linear ◦ SoftMax ◦ Linear− ◦ LReLU ◦ Linear−

Yury Gorishniy et al, On Embeddings for Numerical Features in Tabular Deep Learning

Datasets details

Table 10. Deteails on datasets, used for experiments

Abbr	Name	# Train	# Validation	# Test	# Num	# Cat	Task type	Batch size
GE	Gesture Phase	6318	1580	1975	32	0	Multiclass	128
CH	Churn Modelling	6400	1600	2000	10	1	Binclass	128
EY	Eye Movements	6998	1750	2188	26	0	Multiclass	128
CA	California Housing	13209	3303	4128	8	0	Regression	256
HO	House 16H	14581	3646	4557	16	0	Regression	256
AD	Adult	26048	6513	16281	6	8	Binclass	256
OT	Otto Group Products	39601	9901	12376	93	0	Multiclass	512
HI	Higgs Small	62751	15688	19610	28	0	Binclass	512
FB	Facebook Comments Volume	157638	19722	19720	50	1	Regression	512
SA	Santander Customer Transactions	128000	32000	40000	200	0	Binclass	1024
CO	Covertype	371847	92962	116203	54	0	Multiclass	1024
MI	MSLR-WEB10K (Fold 1)	723412	235259	241521	136	0	Regression	1024

Gorishniy et al. (2022)